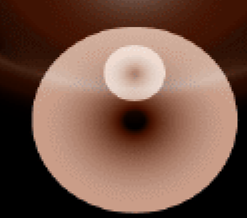
普通高等教育“十一五”国家级规划教材

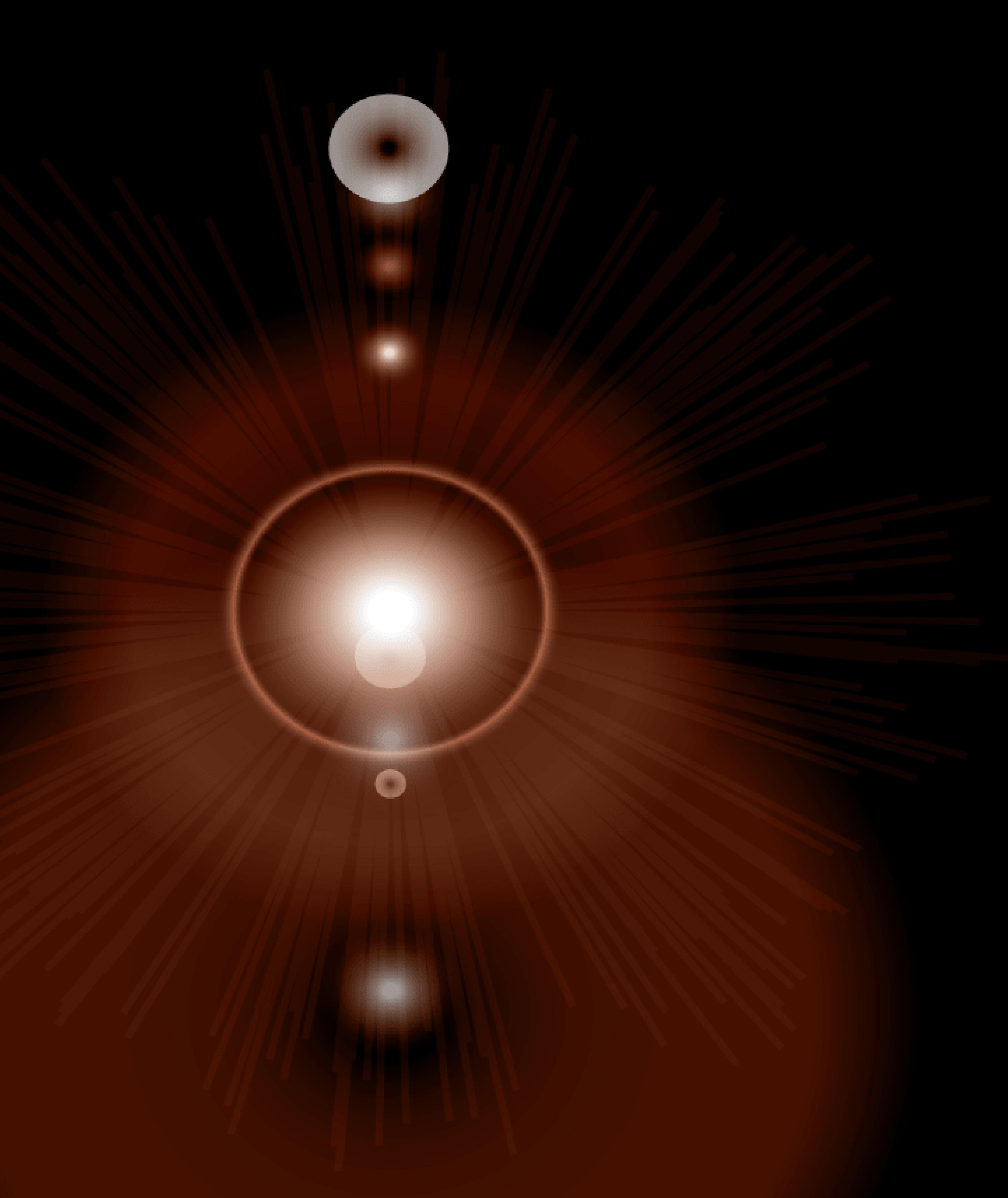


王颖玲 编著

基于Struts和Hibernate技术的Web开发应用

21世纪计算机科学与技术实践型教程

丛书主编 陈明



清华大学出版社

21世纪计算机科学与技术实践型教程

基于Struts和Hibernate技术的Web开发应用

王颖玲 编著

清华大学出版社

北 京

内 容 简 介

本书介绍了开发轻量级 J2EE Web 应用的流行框架 SSH 中的 Struts 和 Hibernate 框架，以及在开发工作中用到的 MyEclipse 和 MySQL 软件的常用操作。全书共分 7 章，从架构思想的引入和架构技术的分析开始，逐步讲解各种技术架构的原型实现，并以登录、注册、退出三项基本功能进行架构的实例讲解。最后通过一个完整的案例整合 Struts＋Hibernate 架构，并给出详细的开发步骤分析。

图书在版编目（CIP）数据

基于 Struts 和 Hibernate 技术的 Web 开发应用/王颖玲编著．—北京：清华大学出版社，2011.9
（21 世纪计算机科学与技术实践型教程）
ISBN 978-7-302-26145-2

Ⅰ．①基…　Ⅱ．①王…　Ⅲ．①软件工具－程序设计－高等学校－教材 ②JAVA 语言－程序设计－高等学校－教材　Ⅳ．①TP311.56 ②TP312

中国版本图书馆 CIP 数据核字(2011)第 122622 号

责任编辑：谢　琛　薛　阳
责任校对：时翠兰
责任印制：杨　艳

出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编**：100084
社 总 机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京国马印刷厂
经　销：全国新华书店
开　本：185×260　**印　张**：9.5　**字　数**：223 千字
版　次：2011 年 9 月第 1 版　**印　次**：2011 年 9 月第 1 次印刷
印　数：1～4000
定　价：20.00 元

产品编号：040593-01

《21 世纪计算机科学与技术实践型教程》

序

21 世纪影响世界的三大关键技术：以计算机和网络为代表的信息技术；以基因工程为代表的生命科学和生物技术；以纳米技术为代表的新型材料技术。信息技术居三大关键技术之首。国民经济的发展采取信息化带动现代化的方针，要求在所有领域中迅速推广信息技术，导致需要大量的计算机科学与技术领域的优秀人才。

计算机科学与技术的广泛应用是计算机学科发展的原动力，计算机科学是一门应用科学。因此，计算机学科的优秀人才不仅应具有坚实的科学理论基础，而且更重要的是能将理论与实践相结合，并具有解决实际问题的能力。培养计算机科学与技术的优秀人才是社会的需要、国民经济发展的需要。

制定科学的教学计划对于培养计算机科学与技术人才十分重要，而教材的选择是实施教学计划的一个重要组成部分，《21 世纪计算机科学与技术实践型教程》主要考虑了下述两方面。

一方面，高等学校的计算机科学与技术专业的学生，在学习了基本的必修课和部分选修课程之后，立刻进行计算机应用系统的软件和硬件开发与应用尚存在一些困难，而《21 世纪计算机科学与技术实践型教程》就是为了填补这部分空白。将理论与实际联系起来，使学生不仅学会了计算机科学理论，而且也学会应用这些理论解决实际问题。

另一方面，计算机科学与技术专业的课程内容需要经过实践练习，才能深刻理解和掌握。因此，本套教材增强了实践性、应用性和可理解性，并在体例上做了改进——使用案例说明。

实践型教学占有重要的位置，不仅体现了理论和实践紧密结合的学科特征，而且对于提高学生的综合素质，培养学生的创新精神与实践能力有特殊的作用。因此，研究和撰写实践型教材是必需的，也是十分重要的任务。优秀的教材是保证高水平教学的重要因素，选择水平高、内容新、实践性强的教材可以促进课堂教学质量的快速提升。在教学中，应用实践型教材可以增强学生的认知能力、创新能力、实践能力以及团队协作和交流表达能力。

实践型教材应由教学经验丰富、实际应用经验丰富的教师撰写。此系列教材的作者不但从事多年的计算机教学，而且参加并完成了多项计算机类的科研项目，他们把积累的经验、知识、智慧、素质融合于教材中，奉献给计算机科学与技术的教学。

我们在组织本系列教材过程中，虽然经过了详细的思考和讨论，但毕竟是初步的尝试，不完善甚至缺陷不可避免，敬请读者指正。

本系列教材主编　陈明

2005 年 1 月于北京

前　　言

为了帮助众多初学者快速掌握轻量级 J2EE 的开发方法，笔者精心编著了本书。它是笔者在项目实战中的经验总结，也是笔者在 Struts/Hibernate 技术的高职教学生涯中根据学生的学习规律和讲义整理而成的。本书根据读者的学习规律，首先通过实例介绍基本概念和基本操作，然后在读者掌握了这些基本概念和操作的基础上，再对内容进行深入的讲解，严格遵循由浅入深、循序渐进的原则。本书按照掌握 Struts/Hibernate 知识的先后顺序进行编排。本书对于每一个实例，从环境配置开始，到最后的运行都有详尽的介绍，从而使读者很容易就能运行实例，掌握开发方法，并体会到学习的快乐，不断增强学习的动力。

本书从头到尾都是按照读者的学习兴趣安排知识点的。我们本着用到什么介绍什么的原则，介绍了开发基于 Struts 和 Hibernate 框架的应用程序的最小软件集合和操作集合。目的就是以开发为向导介绍技术，而不是为技术而技术。

本书适合的读者

本书介绍了开发轻量级 J2EE Web 应用的流行框架 SSH 中的 Struts 和 Hibernate 框架，以及在开发工作中用到的 MyEclipse 和 MySQL 软件的常用操作。使用本书的读者需要具备 Java Web 应用开发的基本技能。本书适合用作 Struts 和 Hibernate 框架的入门书籍。

通过对本书的学习，读者可以掌握 J2EE Web 开发技术，同时通过本书的实战演练，能够积累一定的实际开发经验和技巧，掌握 J2EE Web 开发的思路，为以后的提高打下坚实的基础。

本书可以作为计算机实践型课程的教材，也可以作为 IT 培训机构的教材或是自学材料。

本书特点

✍ 讲解通俗易懂：本书在行文中追求朴实易懂，语言尽量简练，以易读性为第一要务。在编写时充分站在读者的角度描述问题，对于每一个案例，给出全面详尽的步骤分析和操作说明。

✍ 结构主次分明：本书着重讲解开发中常用的技术和工具，使得读者在学习中首先掌握最关键的开发技术，而不用为技术难题所困扰。当读者逐步熟悉开发所使用的常用技术和工具之后，通过进一步的研究学习将很容易地进行技术的升级，并解决开发中遇到的难题。

✍ 由浅入深：为了让读者能很快地进行 J2EE Web 应用开发，每一章都从一个简单的应用示例入手，让读者快速了解本章工作内容，然后再详细讲解本章涉及的基本原理和知识。最后再通过一个详细的示例来巩固学习成果。这种学习过程适合初学者的接受规律。

✍ 实战性强：本书遵循面向工作过程的思想编著而成，在理论部分过后，提供了一个具有原型性的网上购物子系统的需求，以供读者进行练习，融会贯通前面各个章节的内容，从而使读者可以通过本书的学习快速进行实战项目的开发。

本书的组织结构和主要内容

本书内容总体上分为 7 章，从架构思想的引入和架构技术的分析开始，逐步讲解各种技术架构的原型实现，并以登录、注册、退出三项基本功能进行架构的实例讲解。最后通过一个完整的案例整合 Struts＋Hibernate 架构，并给出详细的开发步骤分析。各部分的内容如下：

1. J2EE Web 架构基础篇：开发基于 JSP、JavaBean、Servlet 技术的 MVC 模式的原型系统。

2. Struts 架构篇：开发 Struts 架构原型。

(1) 通过原型系统的 Struts 框架开发，讲解 Struts 原理和核心组件 Action 和 ActionForm。

(2) 通过原型系统的优化，讲解 DispatchAction 的应用。

(3) 通过原型系统视图组件的扩展，讲解 Struts 的标签库和国际化处理。

3. Hibernate 架构篇：开发 Struts＋Hibernate 架构原型。

(1) 讲解 Hibernate 的核心技术，包括持久化对象、Hibernate 基础对象、配置文件和集成 Struts 进行 Web 开发的过程。在 Struts 架构基础上增加 Hibernate 支持。

(2) 在 Struts 架构基础上增加 Hibernate 支持，重点关注 Hibernate 查询的使用。

4. 实战篇：开发基于 Struts＋Hibernate 架构的学生成绩与课程管理系统。

(1) 讲解项目需求。

(2) 以功能为单位展开分析。

(3) 以 MVC 思想贯穿讲解架构的整合。

尽管笔者尽了最大努力，本书难免会有不妥之处，欢迎各界专家和读者朋友批评指正。

作　者

2011 年 8 月

目　　录

第 1 章　J2EE 和 MVC …… 1

1.1　J2EE 简介 …… 1

1.1.1　J2EE 是什么 …… 1

1.1.2　J2EE 技术 …… 2

1.1.3　轻量级 J2EE 开发 …… 3

1.2　软件架构 …… 3

1.2.1　MVC 模式 …… 3

1.2.2　*N* 层架构 …… 6

1.3　构建 MVC 应用之登录功能 …… 6

1.3.1　功能需求描述 …… 6

1.3.2　登录功能应用架构分析 …… 7

1.3.3　数据库分析与建立 …… 8

1.3.4　视图层实现 …… 8

1.3.5　控制层实现 …… 12

1.3.6　模型层实现 …… 15

1.3.7　运行 …… 20

1.4　实验与能力拓展 …… 23

第 2 章　Struts 基本原理和应用 …… 25

2.1　Struts 入门 …… 25

2.1.1　Struts 简介 …… 25

2.1.2　Struts 开发环境的配置 …… 26

2.2　Struts 基本原理与核心组件 …… 29

2.2.1　Struts 基本原理 …… 29

2.2.2　struts-config. xml 配置文件 …… 30

2.2.3　Struts 控制器组件 …… 32

2.2.4　Struts 的 FormBean …… 33

2.3 开发基于 Struts 的应用 …… 35
2.3.1 需求说明 …… 35
2.3.2 开发基于 Struts 的用户登录功能 …… 35
2.3.3 结合案例回顾 Struts 原理 …… 42
2.4 实验与能力拓展 …… 44

第 3 章 使用 DispatchAction 优化控制层 …… 45

3.1 默认的 Action 类 …… 45
3.1.1 默认的 Action …… 45
3.1.2 解读 Action 类的 execute()方法 …… 45
3.2 DispatchAction …… 46
3.2.1 使用 DispatchAction 的必要性 …… 46
3.2.2 DispatchAction 的使用 …… 46
3.3 使用 DispatchAction 改进原型系统 …… 47
3.3.1 使用 DispatchAction 为原型系统添加注册功能 …… 47
3.3.2 使用 DispatchAction 为原型系统添加退出功能 …… 51
3.4 实验与能力拓展 …… 52

第 4 章 使用 Struts 标签进行页面处理和国际化 …… 53

4.1 Struts 中国际化处理 …… 53
4.1.1 国际化 …… 53
4.1.2 资源文件 …… 54
4.1.3 国际化处理过程 …… 57
4.2 Struts 视图组件 …… 58
4.2.1 Struts-html …… 58
4.2.2 Struts-bean …… 61
4.2.3 Struts-logic …… 64
4.3 为原型系统添加国际化处理 …… 67
4.4 实验与能力拓展 …… 71

第 5 章 Hibernate 入门 …… 72

5.1 Hibernate 介绍 …… 72
5.1.1 基础理论 …… 72
5.1.2 Hibernate 简介 …… 73
5.2 Hibernate 应用框架 …… 74
5.2.1 Hibernate 体系结构 …… 74
5.2.2 Hibernate 核心组件 …… 74

5.3 Hibernate 核心 …… 75
5.3.1 Hibernate 配置文件 …… 75
5.3.2 Hibernate 映射文件 …… 76
5.3.3 Hibernate 运行原理 …… 79
5.4 应用 Hibernate 开发原型中的注册功能 …… 79
5.4.1 Hibernate 应用开发流程 …… 79
5.4.2 Hibernate 开发原型中的注册功能 …… 80
5.5 实验与能力拓展 …… 92

第 6 章 Hibernate 查询 …… 93

6.1 Hibernate 查询介绍 …… 93
6.2 HQL 查询基础 …… 94
6.2.1 from 子查询 …… 94
6.2.2 select 子查询 …… 94
6.2.3 where 子查询 …… 95
6.2.4 order by 子查询 …… 96
6.2.5 统计函数查询 …… 96
6.2.6 联接查询 …… 97
6.3 Criteria 查询 …… 97
6.3.1 Criteria 查询的使用步骤 …… 97
6.3.2 创建 Criteria 查询 …… 98
6.3.3 使用 Restriction 类为查询增加限制 …… 98
6.4 应用 HQL 查询实现原型系统的登录功能 …… 99
6.5 实验与能力拓展 …… 101

第 7 章 项目练手：网上购物子系统 …… 102

7.1 系统概述 …… 102
7.2 系统功能演示 …… 103
7.3 系统设计 …… 105
7.3.1 数据库设计 …… 105
7.3.2 创建数据库 …… 105
7.3.3 目录和包结构 …… 107
7.4 工程准备 …… 107
7.5 工程的设计和实现 …… 108
7.5.1 配置工程的 Struts 和 Hibernate 框架支持 …… 108
7.5.2 为工程添加公共类 …… 108
7.5.3 实现 DAO 模式的公共类 …… 111

7.6 用户管理功能的设计和实现 …… 111
7.6.1 用户管理功能的逻辑设计 …… 111
7.6.2 用户管理功能的模型层实现 …… 112
7.6.3 登录和注册功能的视图层实现 …… 123
7.6.4 用户管理功能的控制层实现 …… 132

第 1 章　J2EE 和 MVC

本章导读

本章主要介绍本书所要使用的软件架构，并基于这种思想，实现了一个用户管理的原型系统，包括登录、注册和退出功能。对于本章思想的理解有助于读者在后续的章节中从整体上把握各种技术框架和应用的实现脉络。

本章主要目的是使读者了解 J2EE 的基本组成，分层的软件设计思想——MVC，以及其应用实现。

本章的介绍是基于读者已经具备了 JSP、Servlet 等 Java Web 开发的基本知识的假设展开的。如果读者还没有具备相应的知识，请参考相关书籍。

工作任务

1. 采用 MVC 架构实现用户的登录功能。
2. 采用 MVC 架构实现用户的注册功能。
3. 采用 MVC 架构实现用户的退出功能。

1.1　J2EE 简介

1.1.1　J2EE 是什么

目前，Java 2 平台有三个版本，分别是适用于小型设备和智能卡的 Java 2 平台 Micro 版（Java 2 Platform MicroEdition，J2ME）、适用于桌面系统的 Java 2 平台标准版（Java 2 Platform Standard Edition，J2SE）、适用于创建服务器应用程序和服务的 Java 2 平台企业版（Java 2 Platform Enterprise Edition，J2EE）。

J2EE 是一种利用 Java 2 平台来简化企业解决方案的开发、部署和管理相关的复杂问题的体系结构。J2EE 技术的基础就是核心 Java 平台或 Java 2 平台的标准版，J2EE 不仅巩固了标准版中的许多优点，例如“编写一次、随处运行”的特性、方便存取数据库的 JDBC API、CORBA 技术以及能够在 Internet 应用中保护数据的安全模式等，同时还提供了对 EJB（Enterprise JavaBean）、Java Servlets API、JSP（Java Server Pages）以及 XML 技术的全面支持。其最终目的就是成为一个能够使企业开发者大幅缩短投放市场时间的

体系结构。

通过提供统一的开发平台,J2EE 降低了开发多层应用的费用和复杂性,同时提供了对现有应用程序集成的强有力支持,完全支持 Enterprise JavaBean,有良好的向导支持打包和部署应用,添加目录支持,增强了安全机制,提高了性能。通过图 1-1 可以了解 J2EE 平台中的主要容器。

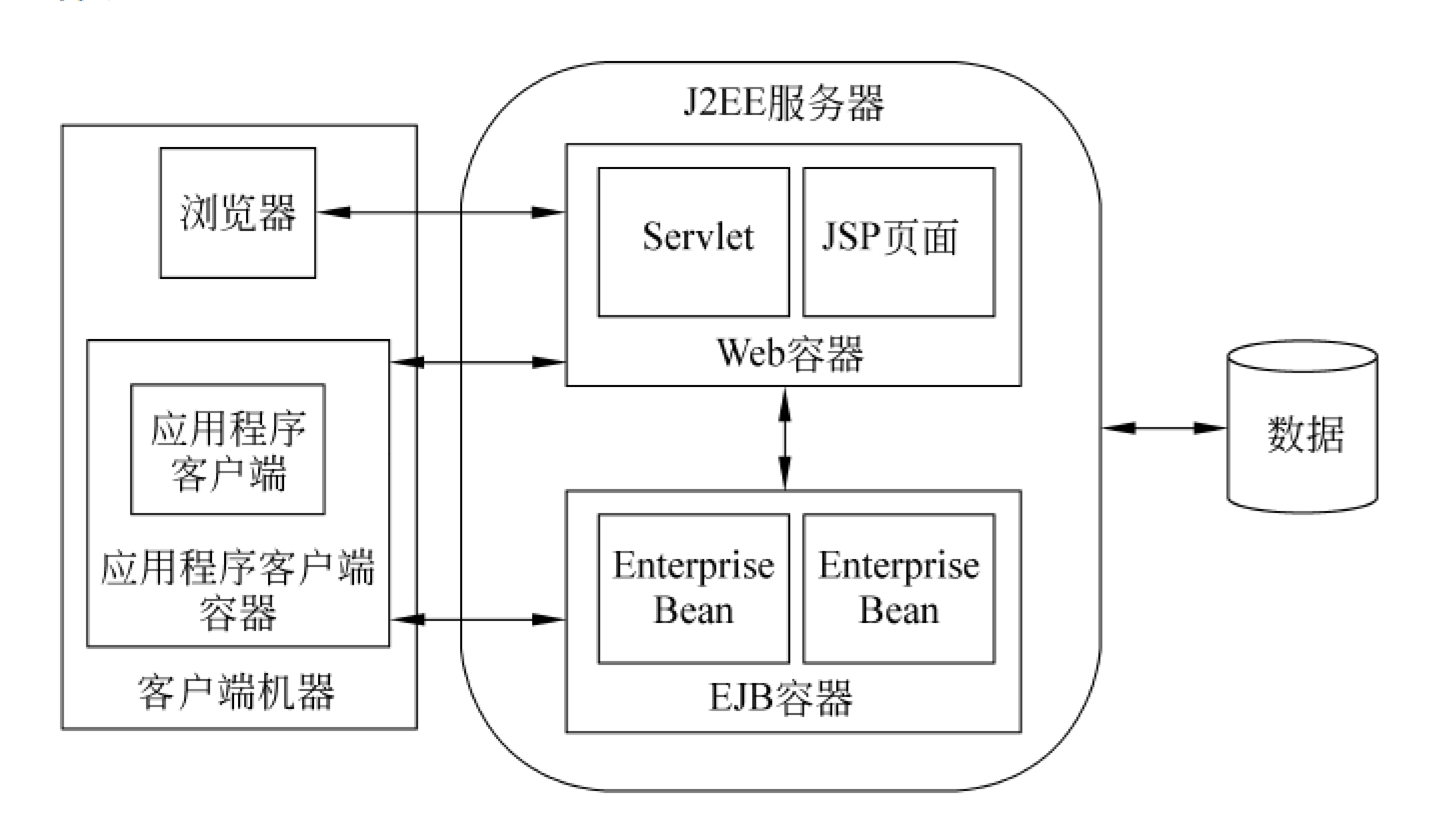


图 1-1 J2EE 框架简图

采用了 J2EE 后的 B/S 结构如图 1-2 所示。

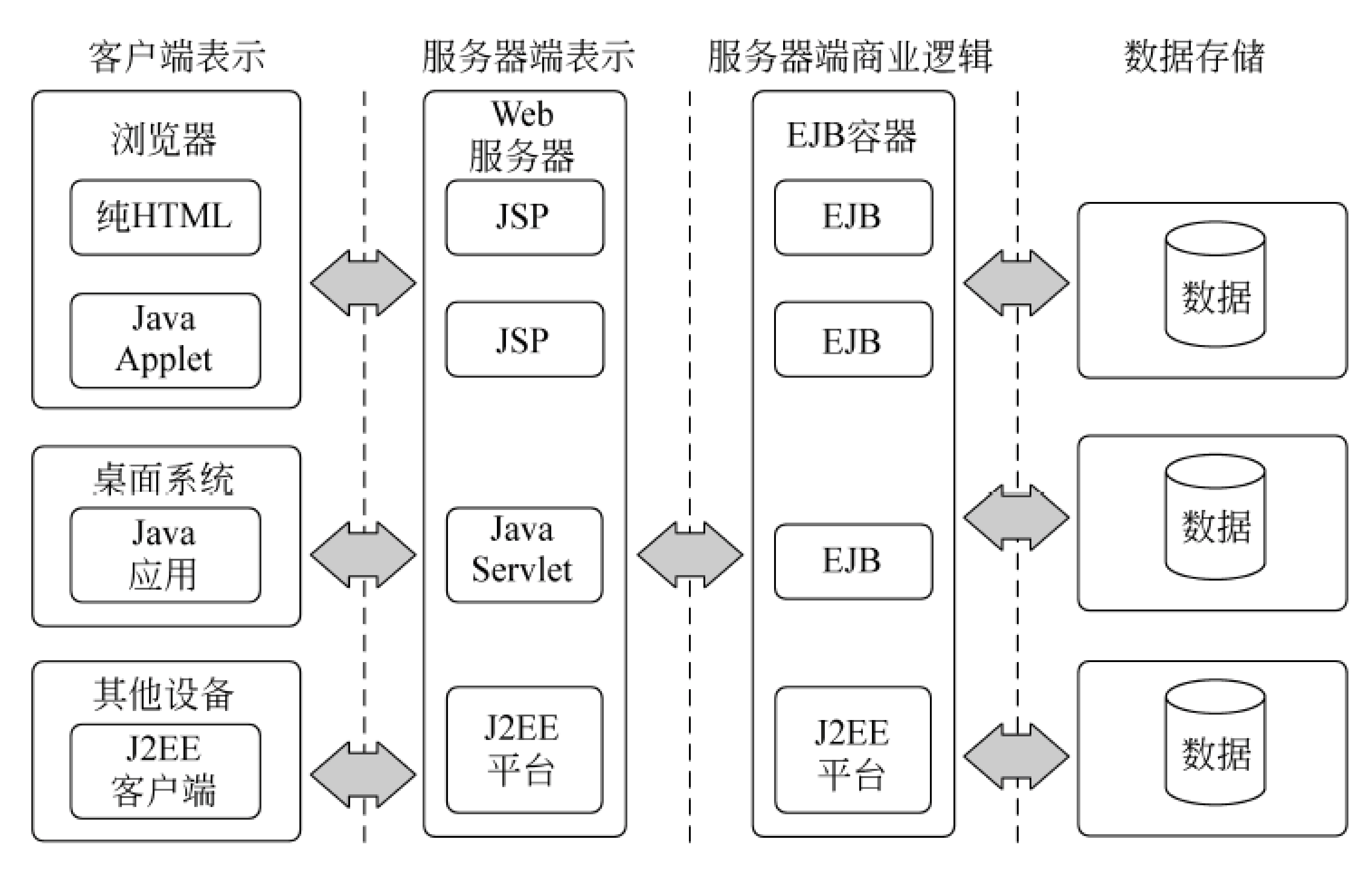


图 1-2 J2EE 应用层次图

通过图 1-1 和图 1-2 可以看出,J2EE 平台开发出来的 Web 应用是采用封装思想对功能分类分层的。通过这种做法,可以提高应用程序的内聚性,降低耦合度,提高组件的可重用性。J2EE 平台支持的应用开发可以降低后期的维护代价,增强应用的可扩展性。

1.1.2 J2EE 技术

J2EE 定义了一个框架和相关的规范,而实现这个框架的具体工具需要第三方厂商来

完成，不同厂商提供的不同工具的实现方式也是不一样的，比如 Tomcat、WebSphere 和 WebLogic 等，它们包含了 J2EE 定义的若干种技术规范。J2EE 核心技术及其应用如图 1-3 所示。

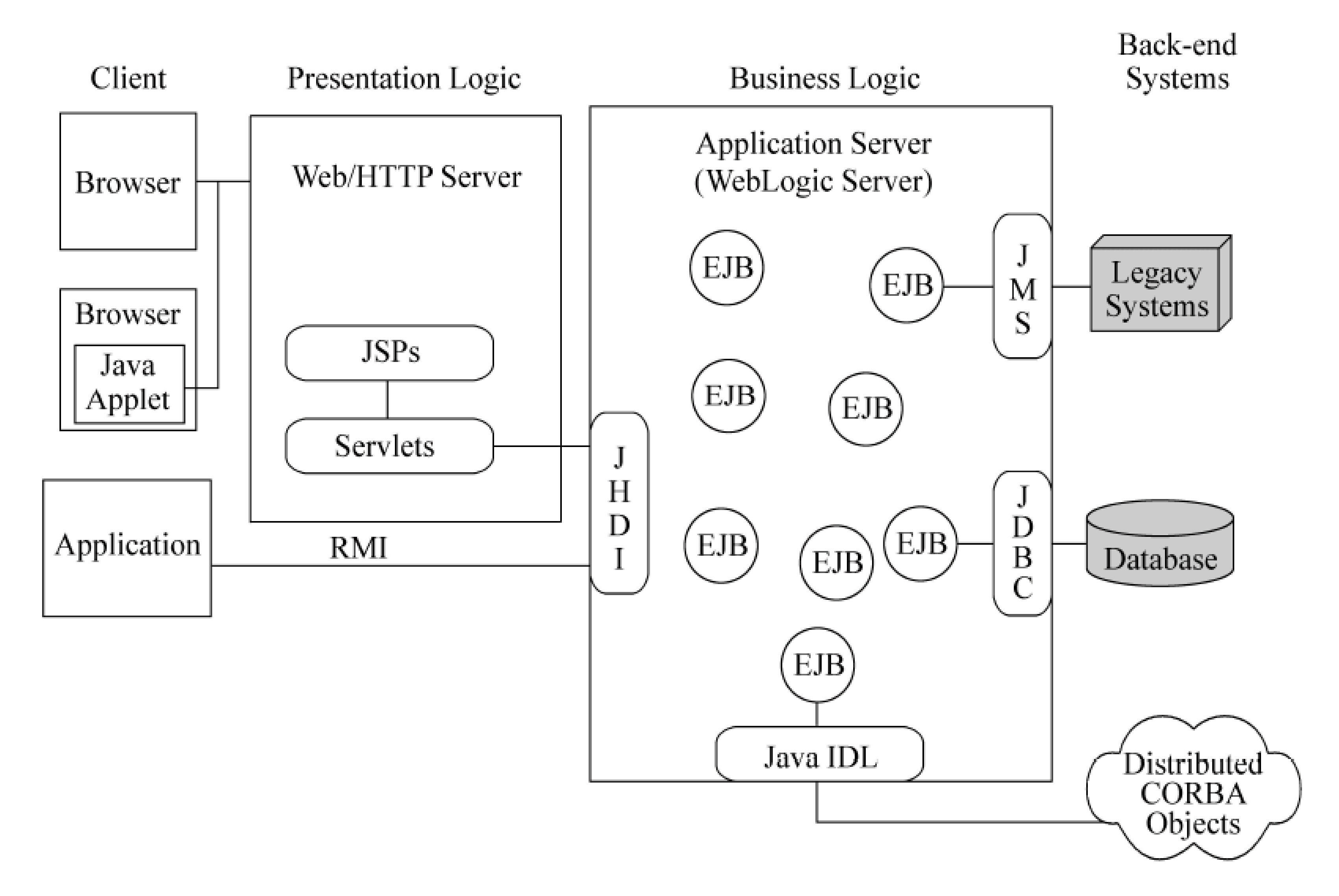


图 1-3　J2EE 核心技术图

1.1.3　轻量级 J2EE 开发

本书不可能对 J2EE 的每一种技术进行详细的介绍，也不可能使读者通过本书能成为 J2EE 专家。本书仅介绍开发 J2EE 应用的两种架构 Struts 和 Hibernate，而不涉及 EJB 等技术。这里还假定使用本书的读者已经掌握了进行 Java Web 开发所必需的 JSP、Servlet 技术。

通常，将不使用 EJB 进行的 J2EE Web 应用开发称为轻量级 J2EE 开发。

1.2　软 件 架 构

1.2.1　MVC 模式

软件架构(software architecture)用于指导大型软件系统各个方面的设计。软件架构是一个半成品，有利于降低开发和维护的成本，使得工作易于开展(就像简历模板、书信格式一样可以给人的工作带来便利)，尤其是对新接手的开发人员来讲。

MVC 架构就是能带来上述便利的一种模式。那么 MVC 究竟是什么呢?

MVC 的英文全称是 Model-View-Controller，中文的意思是“模型-视图-控制器”。MVC 模式起源于 Smalltalk 语言，它是 Xerox PARC 在 20 世纪 80 年代为编程语言 Smalltalk-80 发明的一种软件设计模式。MVC 模式的结构由以下三个部分组成：模型

(Model)、视图(View)、控制器(Controller)。有时也称MVC模式为三层架构。

要想理解并有效地使用MVC,必须分层理解MVC,还必须理解该架构中的三个部分之间是如何通信和合作的。下面通过图1-4来初步理解MVC三层架构和Web应用的关系。

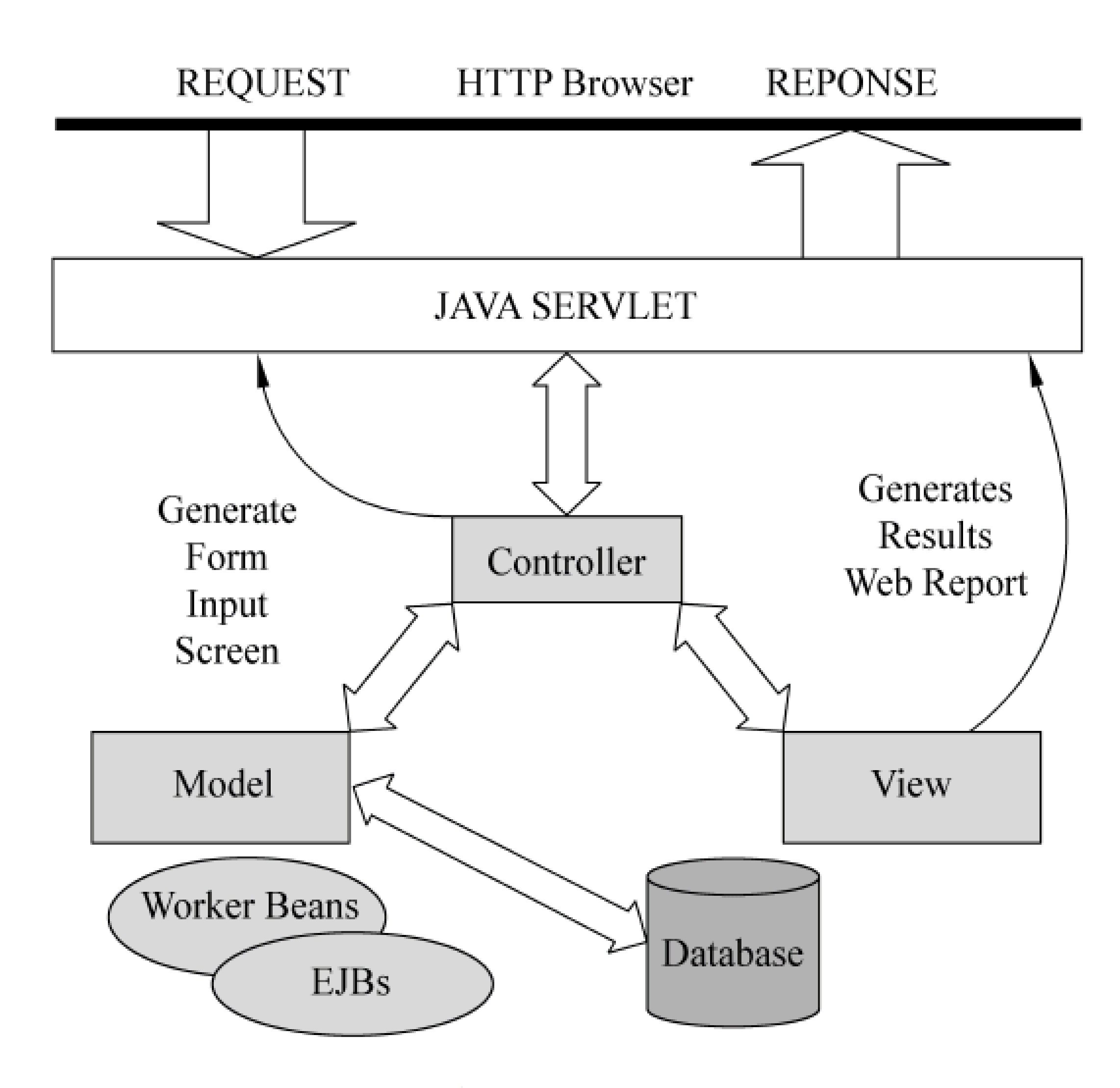


图1-4　Web应用中的MVC之Model 2

按照模型层、视图层、控制层进行分解,从而使得整个系统责任明确、接口清晰,各项开发可以同步进行,加快了设计开发的过程。下面分别从几个方面或角度理解MVC的Model2模式在Web应用中的运行。

1. 整体关系

各个模块是通过明确定义的接口通信合作的。在一个Web应用中,模型、视图和控制器的设计和实现是独立的,不同分工的人员按照统一的设计接口进行各个部分的实现,然后进行集成。

(1) 浏览器(HTTP Browser)负责显示视图层(View)的各个视图,是客户端和系统交互的唯一介质。

(2) 客户和Web应用的交互,即请求的接收和响应的返回都通过控制器进行。而控制器在Model2模式中一般由Servlet实现。

(3) 后台数据库(Database)中存放Web应用的必要数据,但是,数据的处理只通过模型(Model)进行。

(4) 所有经过控制器传递过来的请求或经由Model产生的处理结果都由控制器来协调,在模型与合适的视图之间进行分发。

2. MVC的各个元素

(1) **模型层**这里是指业务逻辑的处理和数据的存储。它分为两类模型:业务逻辑模

型和数据模型。模型接收通过控制器(在Web应用中通常由Servlet担当控制器的角色)转发来的来自视图的请求数据,并返回最终的处理结果。MVC并没有提供模型的设计方法,而只告诉开发人员应该如何组织管理这些模型,以便于模型重构和提高重用性。**数据模型**是指对数据的持久化,它实现了对视图和模型之间交互的支持。实现是把"做什么(业务处理)"和"怎么做(业务实体)"分离,这样可以实现业务逻辑的重用。对一个开发者来说,就可以专注于业务模型的设计。

(2) **视图层**主要用来展现用户所需的数据,它是用户和系统进行交互的界面。这部分工作可以由美工人员进行开发和维护,一般可以采用HTML页面、XML、Servlet和Applet等技术。一般来说,视图只接收来自模型的数据并显示给用户,然后将用户界面的输入数据和请求传递给控制。MVC设计模式对于视图的处理仅限于视图上数据的采集和处理及用户的请求,而不包括在视图上的业务流程的处理,业务流程的处理和状态的改变则交给模型层处理。

(3) **控制层**就是一个分发器。选择什么样的模型处理请求,根据模型处理结果选择什么样的视图,可以完成什么样的用户请求都由控制层决定。控制层就像一个中转站,它从用户那里接受请求,并根据用户的请求,将模型与视图匹配在一起,共同完成用户的请求。

通过将模型、视图和控制器分离,使得一个模型可对应多个视图,一个视图也可以对应多个模型。如果用户通过某个视图的控制器改变了模型的数据,所有其他依赖于这些数据的视图都将反映出这些变化。因此,无论何时发生了何种数据变化,控制器都会将变化通知所有的视图,导致数据的更新。

模型、视图、控制器三者之间的关系和各自的主要功能,即MVC模式的功能如图1-5所示。

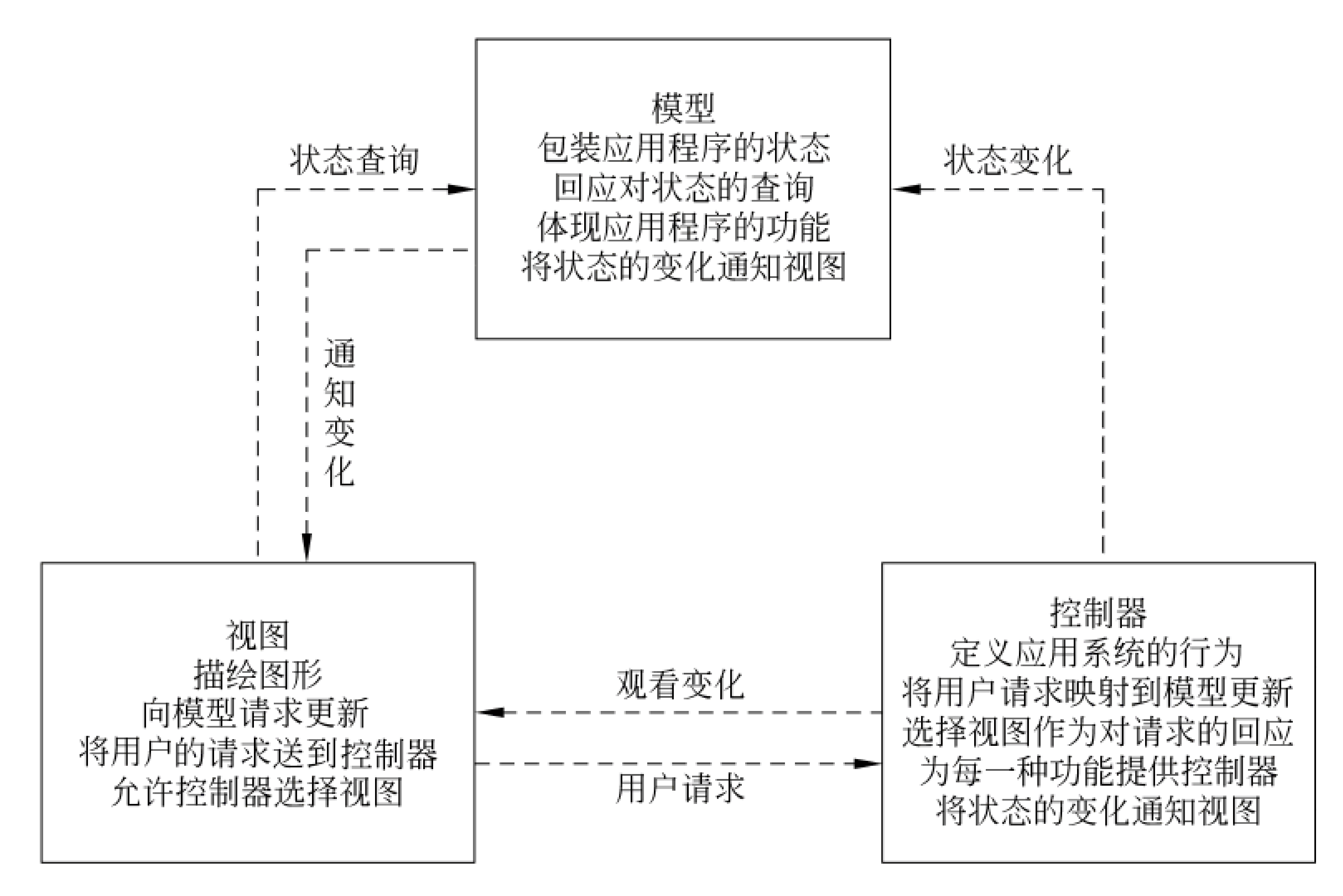


图1-5　MVC模式的功能示意图

1.2.2 N 层架构

三层 MVC 架构分离表示和业务逻辑，给工作的展开和后期的维护带来了极大的便利，提高了工作效率，但是，在实际项目工作中，往往会对三层 MVC 结构做进一步的分层处理，来满足不同复杂程度的项目施工要求。因为扩展层次的不确定性，人们通常称多于三层架构的软件架构为 N 层架构或多层架构。

一种常用的 N 层架构是 4 层架构，即表示层(View)、控制层(Controller)、业务逻辑层(BLL)、数据访问层(DAO+Data Object)。实际上是对前述的 MVC 架构中的模型层进行了拆分。

本书中采用的基于 Struts 和 Hibernate 实现的简易 BBS 管理系统就是采用 N 层(4 层)架构实现的。

1.3 构建 MVC 应用之登录功能

1.3.1 功能需求描述

这里选择的案例是大部分 Web 应用系统中都具有的用户登录、注册和退出功能，在本书后面，称这几项功能组成的系统为原型系统。为了理解用户登录和注册的功能要求，下面先从以下几个方面进行分析说明。

1. 用例图

这里要实现的原型系统中的功能包括三个：登录、注册与退出，如图 1-6 所示。

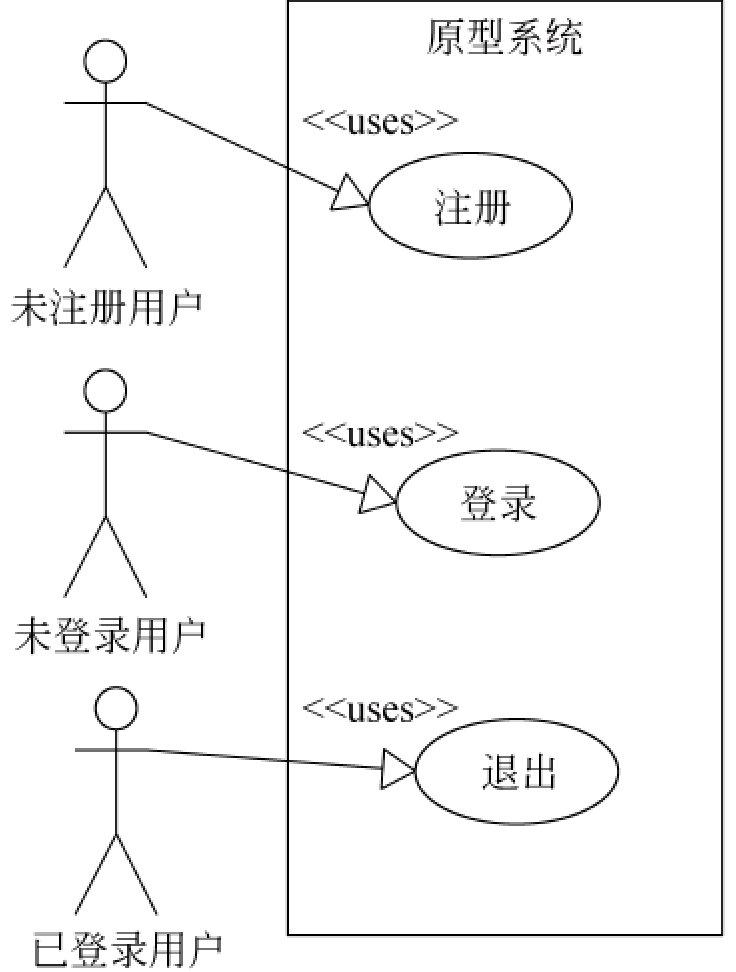


图 1-6 原型系统用例图

系统的用户主要有三类：未注册用户、登录用户和未登录用户。这三类用户和原型系统中的三个功能的关系及功能描述如下。

1）登录功能

只有注册了的用户才能登录进入系统。登录功能界面只需要用户提供用户名和密码即可。如果输入正确的用户名和密码，就可进入系统的欢迎界面，否则转到注册页面。

2）注册功能

未注册的用户必须注册一个用户账号才能执行系统内的特定操作。注册需要提供用户名、密码、确认密码和 E-mail 信息。这里的限制是用户名不能重复。注册成功的用户进入登录页面，可以经由登录功能登入系统。

3）退出功能

用户登录系统之后，即可操作系统。假定在本原型系统上开发了扩展功能，操作完毕之后，经由退出操作可以注销个人登录信息，使用户失去对系统的管理操作权，提高系统

的安全性。退出之后返回到登录页面。

2. 界面流程

用户执行操作期间，能在各个功能界面之间跳转，可以通过图 1-7 清晰地掌握界面之间的转换关系。

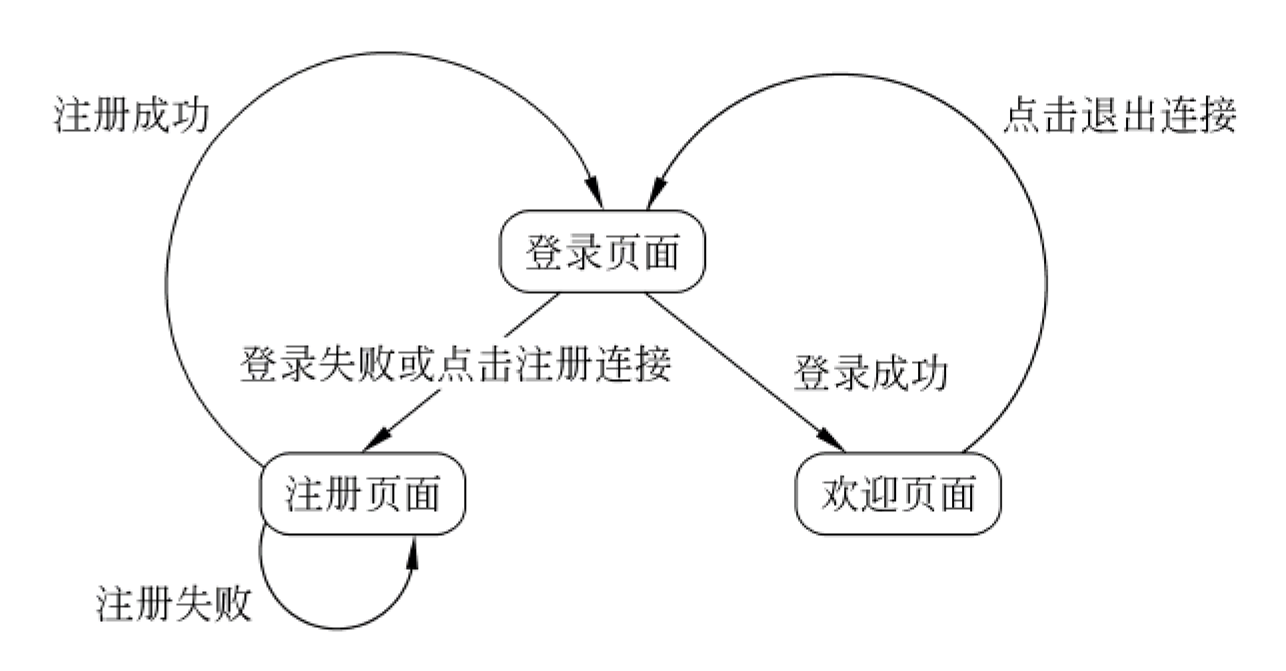


图 1-7 页面状态图

原型系统的首页是登录页面，经由登陆页面，如果登录成功，则进入欢迎页面；否则，进入注册页面。注册成功的用户即可返回登录页面执行登录操作。在欢迎页面的用户可以经由退出操作进入登录页面。

3. 登录功能数据流图

这里以登录功能的实现为案例进行详细的分析和讲解，关于注册和退出功能的实现请参考本书配套的源代码进行理解和练习。

1.3.2 登录功能应用架构分析

这一部分将使用 Model2 的结构对用户登录功能进行实现。实现的过程依照 M-V-C 方式进行。

实现用户登录管理功能的各层主要内容如下：

(1) 模型层：一个 UserHandlerBean 类。该类实现数据库访问功能。其中一个重要的方法是根据用户名和密码查询数据库中是否存在要登录的用户。如果存在则返回真值，否则返回假值。

(2) 视图层：三个 JSP 页面——login. jsp、register. jsp 和 main. jsp。login. jsp 页面上有一个表单，表单中有两个输入框，分别接收用户名和密码，还有一个提交按钮。页面提交信息传递给控制器。main. jsp 作为主页面，当登录成功时转至该页面，显示欢迎信息；否则，转至 register. jsp，等候用户注册。

(3) 控制层：一个 Servlet 的 java 类——UserServlet. java。该类的关键方法是接收用户传递的请求信息，调用模型层的 UserHandlerBean 类的验证方法，根据方法的返回值决定要呈现给用户的视图。

对于用户的登录和注册功能的实现请参考图 1-8。

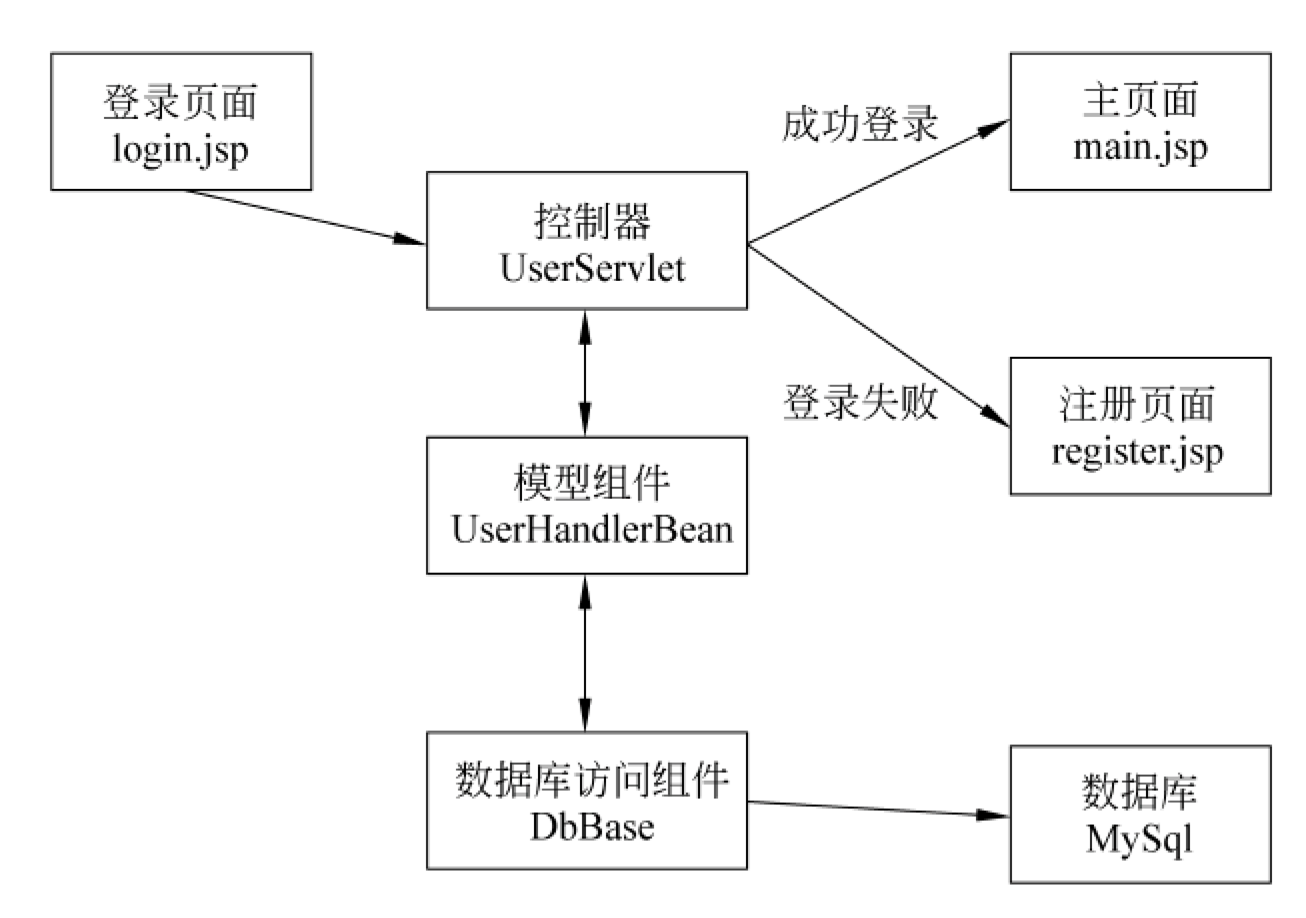


图 1-8 登录功能的简单实现流程图

1.3.3 数据库分析与建立

原型系统中用到的数据库命名为 mvcDB，其中原型系统中的数据库表结构如表 1-1 所示。

表 1-1 tbl_user(用户信息)表结构

列名	类型	长度	允许空	备　注
id	int	4	否	自增长，主键
uname	varchar	50	否	用户名/昵称
upass	varchar	20	否	用户密码
email	varchar	50	是	E-mail 地址

根据上述表结构创建数据库和表的代码如下所示：

```
CREATE DATABASE mvcDB;
USE mvcDB;
CREATE TABLE tbl_user(
id int AUTO_INCREMENT NOT NULL,
uname varchar(50) NOT NULL,
upass varchar(20) NOT NULL,
email varchar(50),
PRIMARY KEY(id)
);
```

1.3.4 视图层实现

本原型系统中有三个视图页面，分别是登录页面 login.jsp、主页面 main.jsp 及注册页面 register.jsp。它们之间的关系如图 1-7 所示。当用户在登录页面输入用户信息之

后，提交信息，系统将检查信息是否正确。如果是正确的注册信息，则跳转到 main. jsp 页面；否则，就进入 register. jsp 页面。

login. jsp 页面的代码如代码清单 1-1 所示。当用户单击登录按钮后，系统把请求提交给一个叫 LoginServlet 的 Servlet，以做进一步的处理。

代码清单 1-1　login. jsp

```
<%@page language="java" import="java.util.*" pageEncoding="GB18030"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>欢迎登录</title>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
    </head>
    <body>
        <form method="post" action="LoginServlet" name="loginForm">
            <p>
                 
            </p>
            <div align="center">
                <blockquote>
                    <blockquote>
                        <p>
                             
                            <input type="hidden" value="login" name=
                            "opType">
                        </p>
                        <p>
                            用户名:
                            <input type="text" maxlength="50" name="uid">
                        </p>
                        <p>
                            密   码:
                            <input type="password" maxlength="20" name="upass">
                        </p>
                        <p>
                             
                            <input type="submit" value="登录" name="btnLogin">
                            <input type="reset" value="撤销" name="btnCancel">
                        </p>
                        <p>
                             
```

```
                        <font color="#ff0000">尚未注册?请来</font>
                        <a href="register.jsp">这里</a>!
                    </p>
                </blockquote>
            </blockquote>
        </div>
    </form>
  </body>
</html>
```

main.jsp 页面的代码如代码清单 1-2 所示。当用户从登录页面进入主页面时,将提示欢迎当前用户的信息和一个退出按钮。

代码清单 1-2 main.jsp

```
<%@page language="java" import="java.util.*" pageEncoding="GB18030"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>My JSP 'main.jsp' starting page</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
  </head>
  <body>
    <div align="center">
      <font size="4"><strong><font color="#008000">
                  ${sessionScope.username}</font>你好!欢迎光临!<br><br>
              <form action="LoginServlet" method="post">
                  <input type="hidden" value="logout" name="opType">
                  <input type="submit" name="submit" value="退出">
              </form><br></strong></font>
    </div>
  </body>
</html>
```

register.jsp 页面的代码如代码清单 1-3 所示。当用户进入注册页面时,需要提供用户 id 和用户密码信息进行注册。注册信息提交给 loginServlet 处理。

代码清单 1-3 register.jsp

```
<%@page language="java" import="java.util.*" pageEncoding="GB18030"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
```

```
    <title>请您注册</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
</head>
<body>
    <form method="post" action="loginServlet" name="loginForm">
        <p>
             
        </p>
        <p>
             
            <input type="hidden" value="register" name="opType">
        </p>
        <p align="center">
            用户名：
            <input type="text" maxlength="50" name="uid">
        </p>
        <div align="center">
        </div>
        <p align="center">
            密   码：
            <input type="password" maxlength="20" name="upass1">
        </p>
        <div align="center">
        </div>
        <p align="center">
            确认密码：
            <input type="password" maxlength="20" name="upass2">
        </p>
        <div align="center">
        </div>
        <p align="center">
            电子邮箱：
            <input type="text" maxlength="50" name="email">
        </p>
        <div align="center">
        </div>
        <p align="center">
             
            <input type="submit" value="注册" name="btnLogin">
            <input type="reset" value="撤销" name="btnCancel">
```

```
                </p>
                <div align="center">
                </div>
                <p align="center">
                     
                </p>
            </form>
        </body>
    </html>
```

代码解读：login. jsp、register. jsp 以及 main. jsp 上的退出操作都提交由同一个 Servlet 进行处理。请注意两个页面中的突出部分代码，从而识别不同的信息来源。

1.3.5 控制层实现

本案例的控制层仅有一个组件，即 UserServlet. java。代码如代码清单 1-4 所示。该组件用 POST 方式获得用户的请求信息，然后根据请求的类型，把请求信息传递给业务层组件 UserHandlerBean 的对应业务方法。如果是登录请求，将由登录业务操作根据信息进行判断，如果信息无误则进入主页面；否则，进入注册页面。注册业务方法的实现作为课后练习，请同学们在读懂登录实现之后自行完成。

代码清单 1-4 UserServlet. java

```
package com.wangyingling.ch1.control;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.wangyingling.ch1.model.UserHandlerBean;

public class UserServlet extends HttpServlet {
    public UserServlet(){
        super();
    }
    public void destroy(){
        super.destroy();            //Just puts "destroy" string in log
        //Put your code here
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
```

```
        //1:获取用户请求信息
        String opType= request.getParameter("opType") != null ? request.getParameter
        ("opType") : "";
        //2:根据请求来源判断应该执行哪个业务逻辑
        if (opType.equals("login")){                    //分支 1: 登录
            Login(request, response);
        } else if (opType.equals("register")){          //分支 2: 执行注册业务
            Register(request, response);
        } else {                                        //分支 3: 执行退出业务
            Logout(request, response);
        }

    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);                       //调用 doGet 方法
    }
    public void init() throws ServletException {
        //Put your code here
    }
    /**
     * 调用 model 层的相关实例方法实现登录业务
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    protected void Login(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
            IOException {
        //获得请求参数
        String uid= request.getParameter("uid");
        String upass= request.getParameter("upass");

        //判断请求参数中是否有空值
        if (uid= = null||upass= = null){
            response.sendRedirect("login.jsp");
            return;
        }

        //进行登录验证
        UserHandlerBean UserHandlerBean= new UserHandlerBean();
        boolean isValid= UserHandlerBean.valid(uid, upass);

        if (isValid){
```

```
            HttpSession session= request.getSession();
            session.setAttribute("username", uid);
            response.sendRedirect("main.jsp");
            return;
        } else {
            response.sendRedirect("login.jsp");
            return;
        }
    }

    /**
     * 调用 model 层实例方法实现退出业务
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    protected void Logout(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        HttpSession session= request.getSession();
        session.removeAttribute("username");
        response.sendRedirect("login.jsp");
    }

    /**
     * 调用 model 层实例方法实现注册业务
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    protected void Register(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //获得请求参数
        String uid= request.getParameter("uid");
        String upass1= request.getParameter("upass1");
        String upass2= request.getParameter("upass2");
        String email= request.getParameter("email");

        //判断请求参数是否有空值
        if (uid==null||upass1==null||upass2==null||!upass1.equals(upass2)){
            response.sendRedirect("register.jsp");
            return;
        }
```

```
            //验证用户名是否已经存在并执行相关业务
            UserHandlerBean UserHandlerBean=new UserHandlerBean();
            boolean isExist=UserHandlerBean.isExist(uid);
            if(!isExist){
                UserHandlerBean.add(uid, upass1, email);
                response.sendRedirect("login.jsp");
                return;
            } else {
                response.sendRedirect("register.jsp");
                return;
            }
        }

    }
```

1.3.6　模型层实现

本案例中用到的模型层组件主要有 UserHandlerBean.java，代码如代码清单 1-5 所示。它首先从数据库访问组件 DbBase(见代码清单 1-6)获得数据库连接，然后执行业务方法，实现查询功能或注册功能。

代码清单 1-5　UserHandlerBean.java

```
package com.wangyingling.ch1.model;

public class UserHandlerBean extends DbBase{
    /**
     * 验证有效性
     * @param username
     * @param password
     *  @return
     * /
    public boolean valid(String username, String password){
        boolean isValid=false;
        DbBase db=new DbBase();
        if(db.createConn()){
            String sql="select * from tbl_user where uname='"+username+"' and
            upass='"+password+"'";
            db.query(sql);
            if(db.next()){
                isValid=true;
            }
            db.closeRs();
            db.closeStm();
```

```
                db.closeConn();
            }
            return isValid;
        }
        /**
         * 验证是否已经存在指定用户名
         * @param username
         * @return
         */
        public boolean isExist(String username){
            boolean isExist=false;
            DbBase db=new DbBase();
            if(db.createConn()){
                String sql="select * from tbl_user where uname='"+username+"'";
                db.query(sql);
                if(db.next()){
                    isExist=true;
                }
                db.closeRs();
                db.closeStm();
                db.closeConn();
            }
            return isExist;
        }
        /**
         * 执行注册功能
         * @param username
         * @param password
         * @param email
         */
        public void add(String username, String password, String email){
            DbBase db=new DbBase();
            if(db.createConn()){
                String sql="insert into tbl_user(uname,upass,email)
                values('"+username+"','"+password+"','"+email+"')";
                db.update(sql);
                db.closeStm();
                db.closeConn();
            }
        }
    }
```

代码清单 1-6 DbBase.java

```
package com.wangyingling.ch1.model;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DbBase {

    private String driverName= "com.mysql.jdbc.Driver";     //驱动程序名

    private String url= "jdbc:mysql://localhost:3306/mvcDB";

    private String userName= "root";                         //数据库用户名

    private String userPassword= "root";                     //数据库用户密码

    private Connection conn= null;

    private Statement stm= null;

    private ResultSet rs= null;

    /**
     * 建立数据库连接
     * 如果成功建立连接,返回 true,否则返回 false
     * @ return
     * /
    public boolean createConn(){
        boolean b= false;
        try {
            Class.forName(driverName).newInstance();
            conn= DriverManager.getConnection(url, userName, userPassword);
            b= true;
        } catch (SQLException e){
        } catch (ClassNotFoundException e){
        } catch (InstantiationException e){
        } catch (IllegalAccessException e){
        }
        return b;
    }

    /**
     * 执行修改数据库操作
```

```
     * @param sql
     * @return
     */
    public boolean update(String sql){
        boolean b=false;
        try {
            stm=conn.createStatement();
            stm.execute(sql);
            b=true;
        } catch (Exception e){
            System.out.println(e.toString());
        }
        return b;
    }

    /**
     * 执行查询操作
     * @param sql
     */
    public void query(String sql){
        try {
            stm=conn.createStatement();
            rs=stm.executeQuery(sql);
        } catch (Exception e){
        }
    }

    public boolean next(){
        boolean b=false;
        try {
            if(rs.next())b=true;
        } catch (Exception e){
        }
        return b;
    }

    public String getValue(String field){
        String value=null;
        try {
            if(rs!=null)value=rs.getString(field);
        } catch (Exception e){
        }
        return value;
    }
```

```
public void closeConn(){
    try {
        if (conn != null)
            conn.close();
    } catch (SQLException e){
    }
}

public void closeStm(){
    try {
        if (stm != null)
            stm.close();
    } catch (SQLException e){
    }
}

public void closeRs(){
    try {
        if (rs != null)
            rs.close();
    } catch (SQLException e){
    }
}

public Connection getConn(){
    return conn;
}

public void setConn(Connection conn){
    this.conn= conn;
}

public String getdriverName(){
    return driverName;
}

public void setdriverName(String driverName){
    this.driverName= driverName;
}

public String getuserPassword(){
    return userPassword;
}
```

```
    public void setuserPassword(String userPassword){
        this.userPassword= userPassword;
    }

    public ResultSet getRs(){
        return rs;
    }

    public void setRs(ResultSet rs){
        this.rs= rs;
    }

    public Statement getStm(){
        return stm;
    }

    public void setStm(Statement stm){
        this.stm= stm;
    }

    public String getUrl(){
        return url;
    }

    public void setUrl(String url){
        this.url= url;
    }

    public String getuserName(){
        return userName;
    }

    public void setuserName(String userName){
        this.userName= userName;
    }
}
```

1.3.7 运行

至此，已经完成了原型系统的全部编码工作，现在可以开始发布并运行该 Web 应用程序查看效果了。

(1) 在 Eclipse 开发环境中选中工程 mvcDemo，即创建的示例工程。

（2）单击工具栏上的发布按钮：。

（3）在弹出的发布对话框中，单击 Add 按钮，如图 1-9 所示。

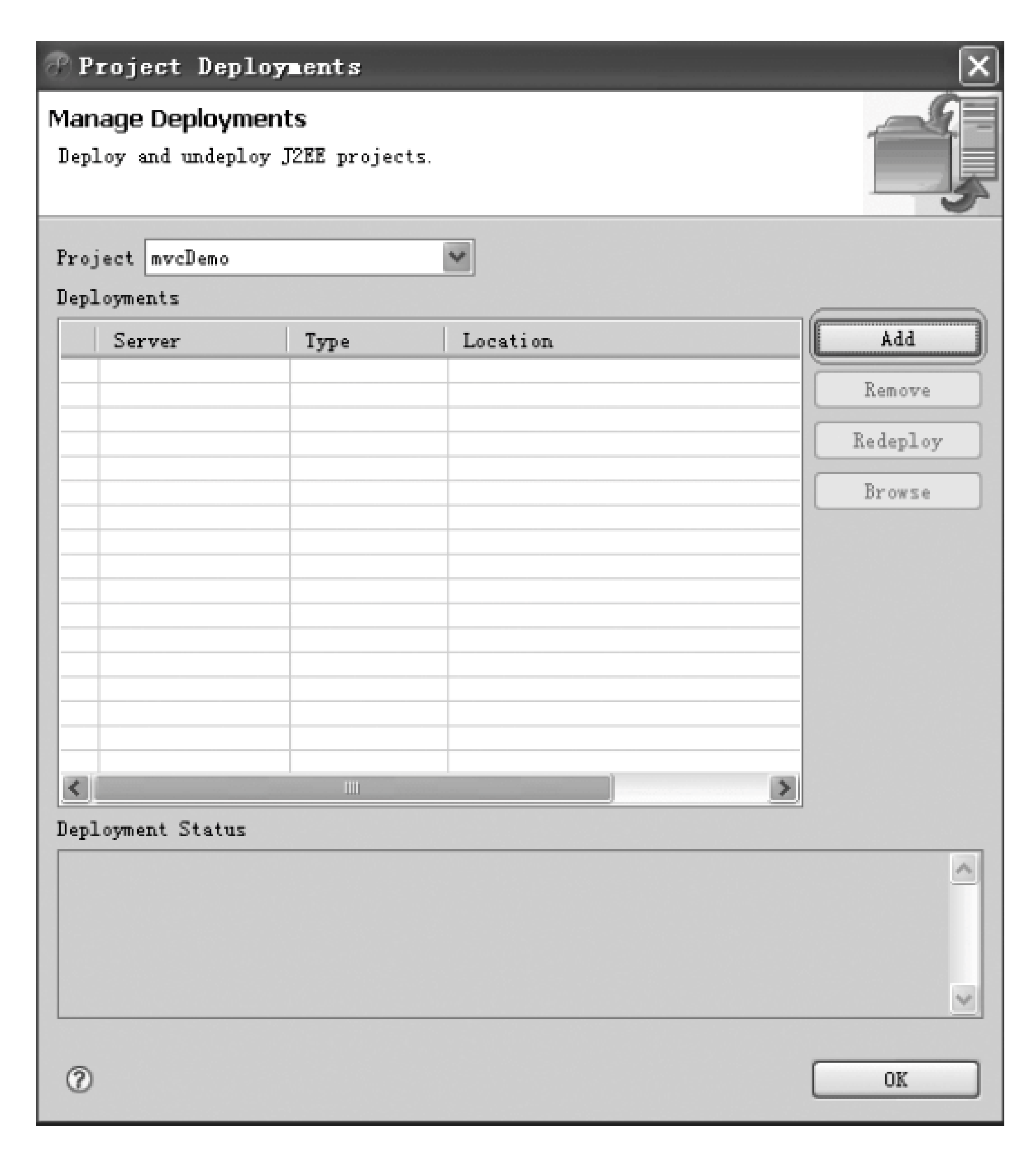


图 1-9 发布对话框

（4）在弹出的选择服务器对话框中选择配置好的 Tomcat5. x 服务器，如图 1-10 所示。

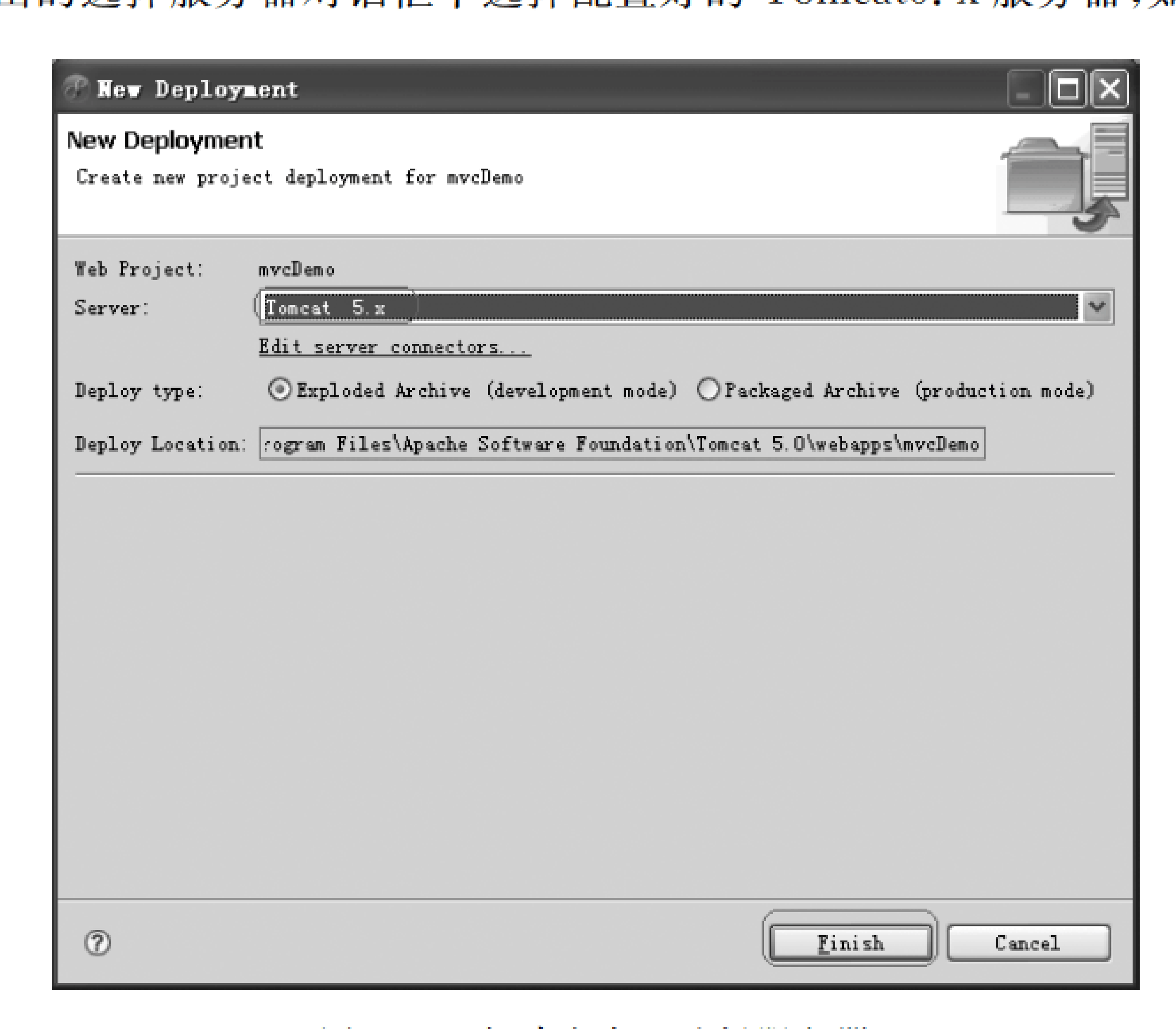


图 1-10 新建发布—选择服务器

（5）单击 Finish 按钮，返回到上一个对话框，此时的对话框如图 1-11 所示，单击 OK 按钮返回，完成发布工作。

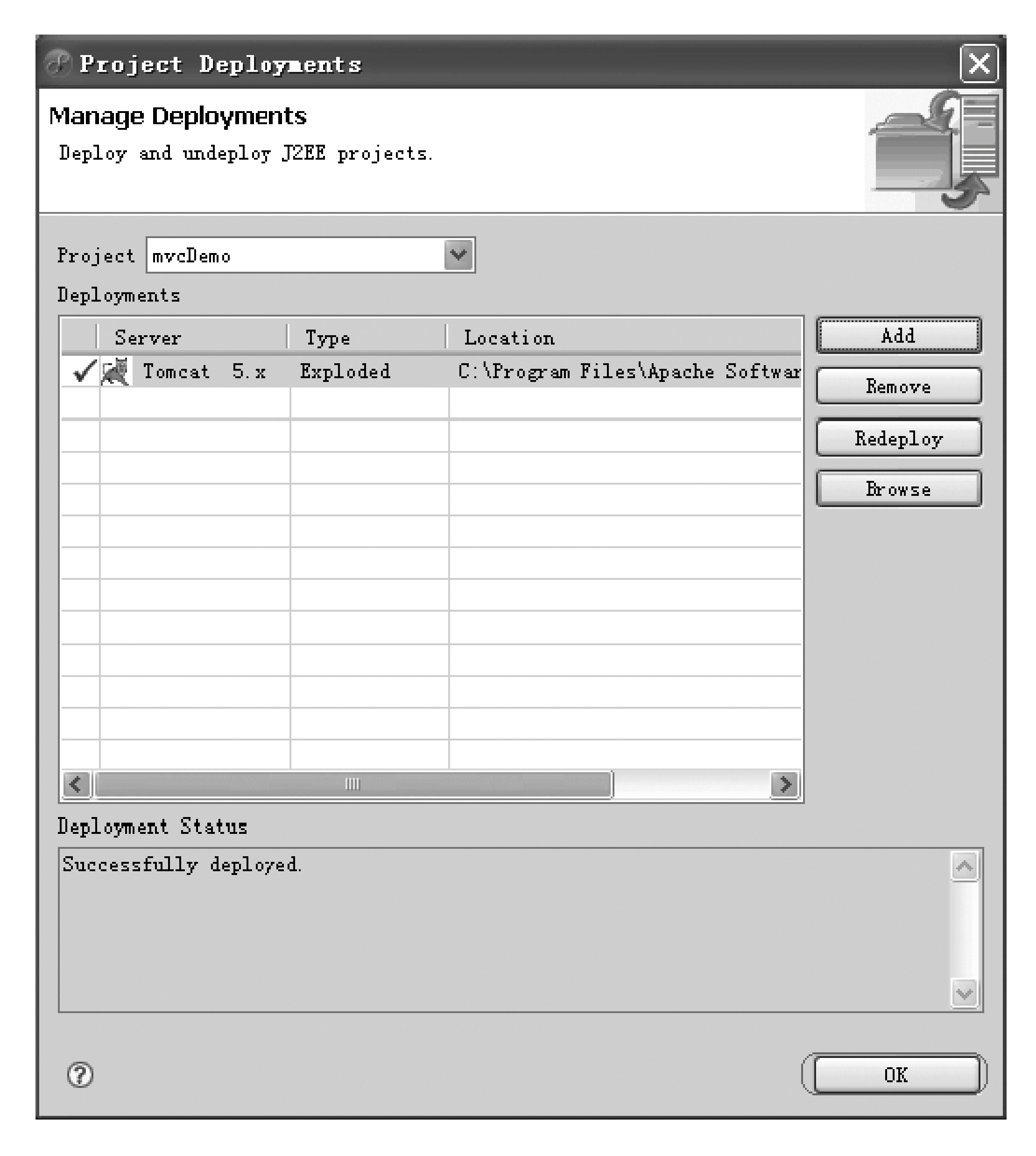


图 1-11 发布项目之后的对话框

（6）现在打开 Servers 视图，并选中 Tomcat 5.x 服务器，单击图右上角的运行或调试按钮，启动服务器，如图 1-12 所示。

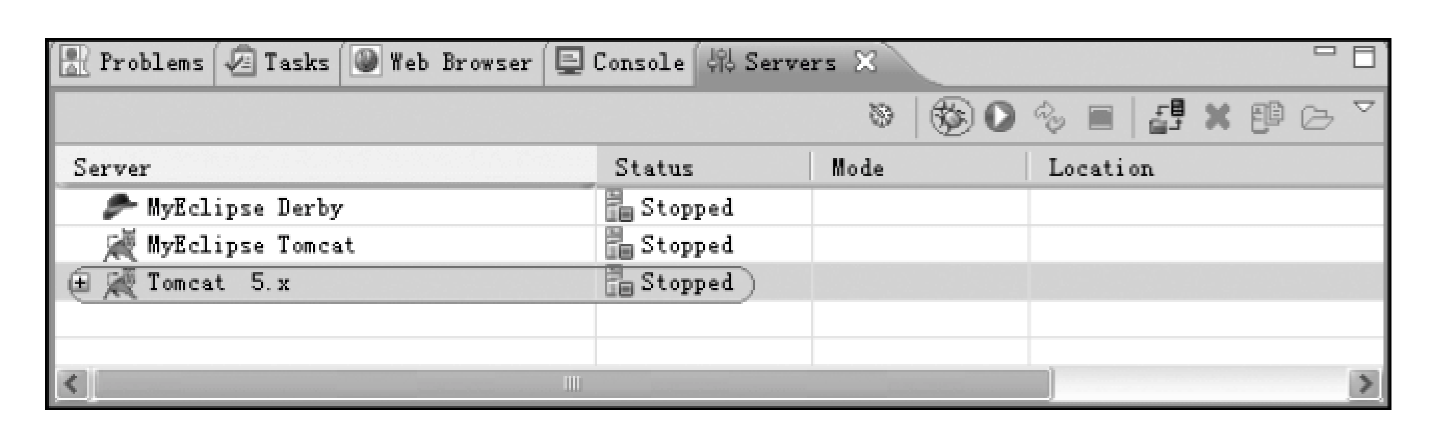


图 1-12 Servers 视图

（7）打开 IE 浏览器，在地址栏中输入地址：http://localhost:8080/mvcDemo/login.jsp，就可以开始运行本案例程序了。可以看到三个视图界面分别如图 1-13～图 1-15 所示。



图 1-13 登录页面

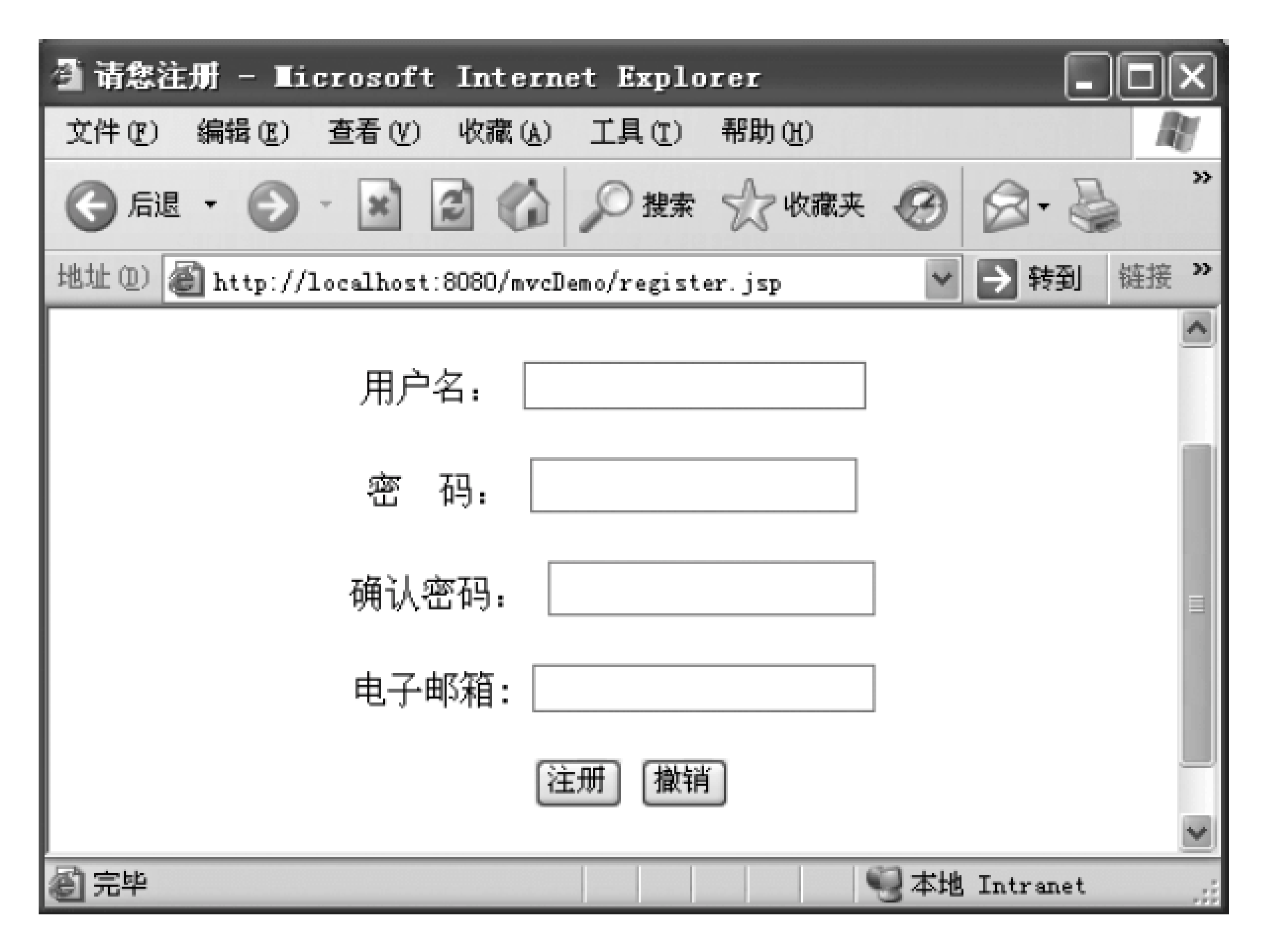


图 1-14　注册页面



图 1-15　欢迎页面

1.4　实验与能力拓展

实验指导：

1. 目标：

a）上机练习完成书中案例中的原型系统并调试运行。

b）通过练习讨论这种 MVC 架构在应用业务逻辑发生改变时，给你的项目维护工作带来了什么样的便利。

c）通过练习理顺 MVC 架构的思想和实施过程。

工具以及配置说明：

d）MyEclipse 7.0

e）MySQL5

f）开发 Web 应用的工具使用若有不明白之处请参考相关书籍。

2. 推荐步骤：

a）创建 mvcDB 数据库

b）新建 Web 工程

c）新建 login.jsp、register.jsp、main.jsp

d）编写 DbBase 的代码

e）编写 UserHandlerBean 的代码

f）编写 UserServlet 的代码

g）运行 Web 程序

3. 运行说明：需要指定 Web 服务器为 Tomcat。其运行步骤请参考上册课本。

能力拓展：

4. 请在完成本教程案例的基础上进行下述扩展：

a）如果登录失败转入注册页面，则在注册页面上显示提示信息“您尚未注册用户名或密码错误，请现在完成注册”，如果从登录页面直接通过超链接进入注册页面则不需要提示信息。

b）如果注册失败则转回注册页面，并提示错误原因的相关信息。

c）请根据您的实际情况对原型系统的模型进行相应修改，使其业务逻辑符合您的实际需要。

5. 请根据上述修改过程，讨论

a）在修改过程中，您的已有项目有哪些改动。

b）MVC 架构和之前您的非架构建设的项目相比，带来了什么便利。

第 2 章　Struts 基本原理和应用

本章导读

本章主要介绍了本书所要使用的 Struts 开源框架。主要内容包括 Struts 框架开发环境的搭建,Struts 应用开发流程以及 Struts 框架的运行原理。在应用上我们实现了基于 Struts 的原型中的用户登录功能,然后又结合该应用对 Struts 原理进行了再次理顺。

本章主要目的是使读者掌握 Struts 的核心原理以及基于 Struts 的应用开发。

工作任务

1. 配置 Struts 开发环境。
2. 实现基于 Struts 的用户管理功能。

2.1　Struts 入门

2.1.1　Struts 简介

Struts 是一个基于 MVC 模式的软件框架,它很好地实现了 MVC 模式。Struts 最早是作为 Apache Jakarta 项目的组成部分问世运作的。Struts 这个名字来源于在建筑中使用的金属架。这个软件框架之所以叫"Struts",是为了提醒我们记住那些支撑我们房屋、桥梁等建筑的框架,这也精彩地解释了在开发 Web 应用的过程中 Struts 所扮演的角色。使用它的目的是为了减少运用 MVC 设计模型来开发 Web 应用的时间。

Struts 作为 MVC 模式的一个优秀实现,在 Web 应用开发中得到了广泛应用。它主要对 MVC 模式中的视图层实现和控制器层实现做了许多工作,方便了程序员的使用,对于模型层,并未做特殊的处理工作,模型层的开发只要遵守 JavaBean 规范等进行即可。

Struts 框架采用了配置文件管理各层之间的通信,这种方式提高了程序的可移植性和跨平台性,也是 Struts 框架的最主要特点。能否很好地掌握和运用这一特点,也成为衡量程序员能否很好地理解和运用 Struts 框架的重要因素。

2.1.2 Struts 开发环境的配置

要进行基于 Struts 的功能开发，就需要添加 Struts 的开发支持环境。所谓的 Struts 支持环境，就是在 Java Web 应用中添加开发 Struts 应用所需要的 jar、tld、xml 等文件，有了这些文件就可以开发 Struts 相关的功能代码。然后进行相应的配置就可以运行 Struts 支持的应用了。

本书采用的开发环境是 MyEclipse + Eclipse，所以，我们需要做的就是使用 MyEclipse 添加 Struts 的开发环境。

> **提示：** 除了使用 MyEclipse 添加 Struts 的开发环境，还可以采用手动方式添加 Struts 的开发环境。具体操作步骤和使用 MyEclipse 添加 Struts 的开发环境一致，只是操作不同。详细操作请参考相关材料。

配置 Struts 应用的开发环境一般需要如下 4 个步骤：

（1）添加 jar 支持包。

（2）添加 tld 标签库。

（3）添加核心配置文件 struts-config.xml，即 Struts 配置文件。

（4）在 web.xml 中添加 Struts 的 ActionServlet 标签。

在 MyEclipse 中配置 Struts 开发环境时，上述 4 个步骤是在 MyEclipse 提供的 Struts 配置向导下一步一步完成的，只需要在相应的界面中完成参数配置即可。MyEclipse 配置向导配置 Struts 开发环境的详细步骤如下。

在配置完成之前，我们假定要开发的原型系统工程名为 strutsDemo，所以首先新建一个 Web 工程，工程名为 strutsDemo。新建的 Web 工程的目录如图 2-1 所示。其中 web.xml 是 Web 工程的配置文件，其内容如代码清单 2-1 所示。

```
strutsDemo
  src
  JRE System Library [jdk1.5]
  J2EE 1.4 Libraries
  Referenced Libraries
  WebRoot
    META-INF
    WEB-INF
      lib
      web.xml
    index.jsp
```

图 2-1 简单的 Web 工程目录结构

代码清单 2-1 没有添加 Struts 开发支持的 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <welcome-file-list><!-首页设置-->
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

1. 使用 MyEclipse 添加 Struts 开发环境

- 首先选中要进行 Struts 开发的项目，然后执行右击菜单操作，如图 2-2 所示。

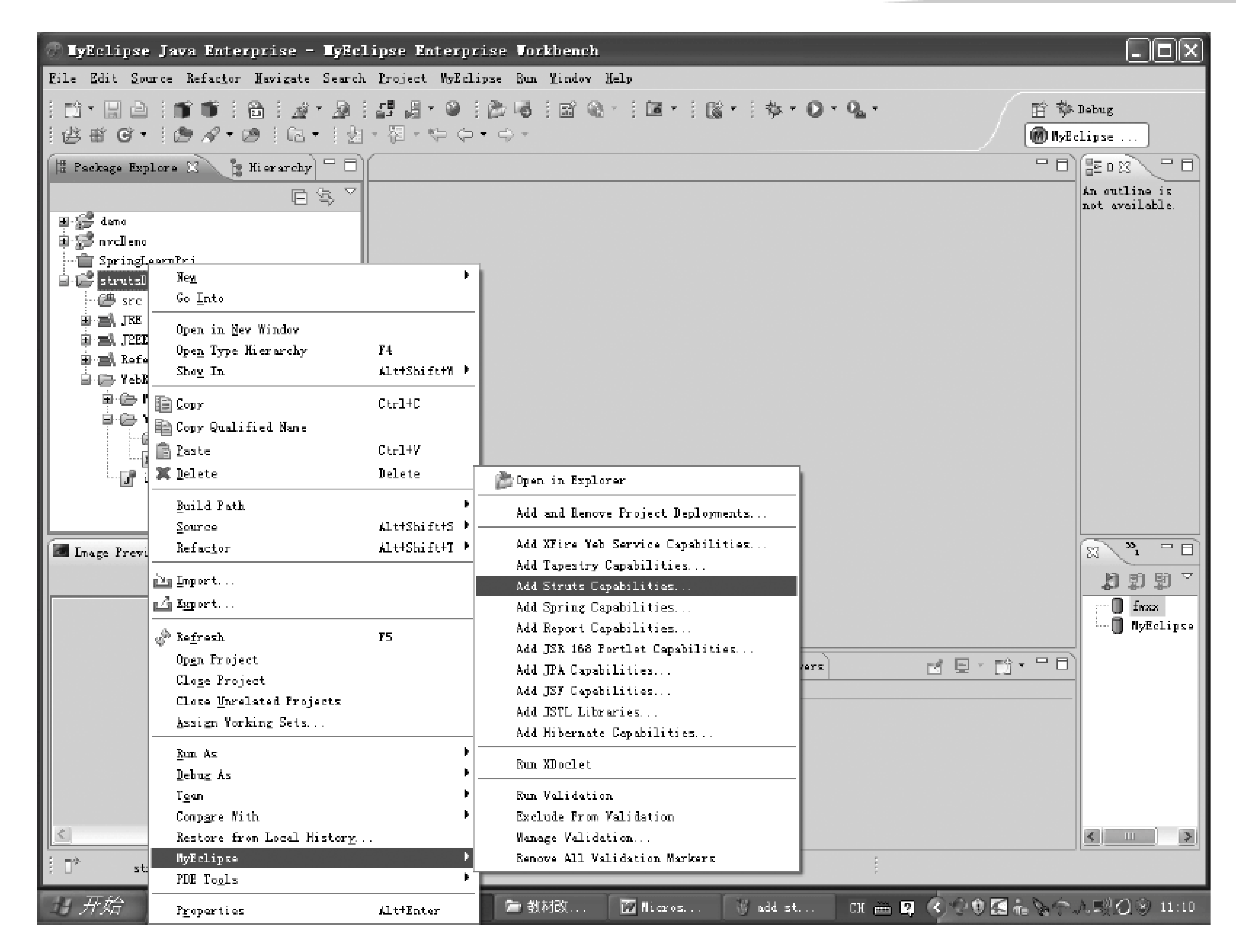


图 2-2　添加 Struts 支持

- 在弹出的对话框中执行 Struts 开发环境的配置操作如图 2-3 所示。

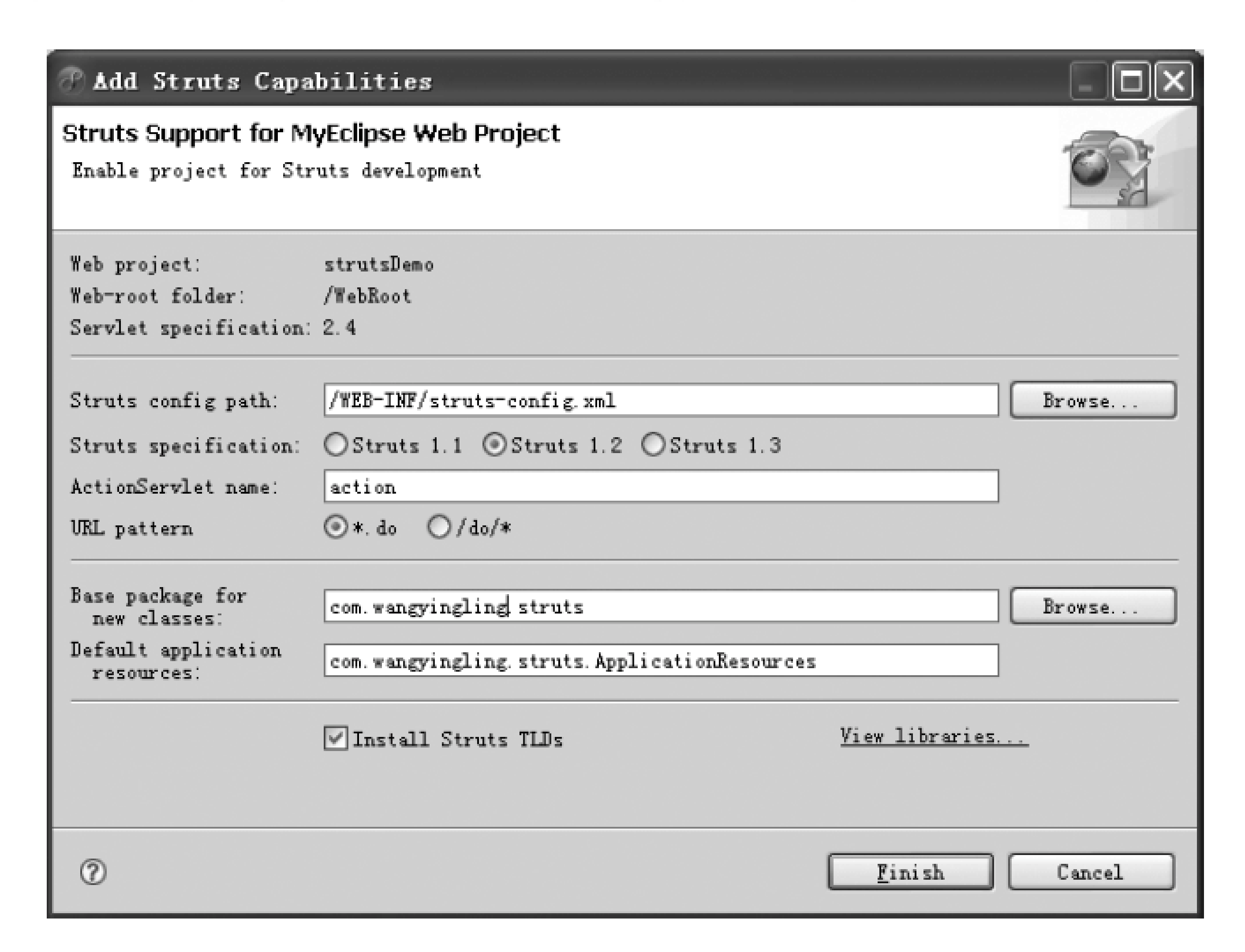


图 2-3　配置 Struts 参数

提示：配置文件的名称默认是 struts-config.xml，但是可以更改。在这里的 Struts config path 的文本框处可以修改配置文件名和路径。

也可以稍后在 web.xml 文件中进行相应的修改来完成配置。

- 执行完 Struts 开发环境配置的 Web 项目的目录结构和主要内容如图 2-4 所示。

配置完成 Struts 开发环境之后的 Web 应用的配置文件的内容如代码清单 2-2 所示，其内容变化正是配置 Struts 开发环境所必需的。

```
strutsDemo
  src
  JRE System Library [jdk1.5]
  J2EE 1.4 Libraries
  Referenced Libraries
  Struts 1.2 Libraries
  WebRoot
    META-INF
    WEB-INF
      lib
      struts-bean.tld
      struts-config.xml
      struts-html.tld
      struts-logic.tld
      struts-nested.tld
      struts-tiles.tld
      validator-rules.xml
      web.xml
    index.jsp
```

图 2-4 空 Struts 项目的目录结构

代码清单 2-2 web. xml

```
< ?xml version= "1.0" encoding= "UTF- 8"?>
< web- app xmlns= "http://java.sun.com/xml/ns/j2ee"
xmlns:xsi= "http://www.w3.org/2001/XMLSchema-
instance"
version= "2.4" xsi:schemaLocation=
"http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web- app_2_4.xsd">
  < servlet>
    < servlet- name> action< /servlet- name>
< servlet- class> org.apache.struts.action.ActionServlet< /servlet- class>
    < init- param>
      < param- name> config< /param- name>
      < param- value> /WEB- INF/struts- config.xml< /param- value>
    < /init- param>
    < init- param>
      < param- name> debug< /param- name>
      < param- value> 3< /param- value>
    < /init- param>
    < init- param>
      < param- name> detail< /param- name>
      < param- value> 3< /param- value>
    < /init- param>
    < load- on- startup> 0< /load- on- startup>
  < /servlet>
  < servlet- mapping>
    < servlet- name> action< /servlet- name>
    < url- pattern> * .do< /url- pattern>
  < /servlet- mapping>
  < welcome- file- list>
    < welcome- file> index.jsp< /welcome- file>
  < /welcome- file- list>
< /web- app>
```

2. 预览结果：Struts 的配置文件 struts-config. xml

目录结构中的 struts-config. xml 文件是 Struts 开发支持的关键配置文件，所有 Struts 应用的配置都在该文件中有所体现。现在，该文件的内容如代码清单 2-3 所示。

代码清单 2-3　空 Struts 应用的 struts-config. xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config_
1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans />
  <global-exceptions />
  <global-forwards />
  <action-mappings />
  <message-resources parameter="com.wangyingling.struts.ApplicationResources" />
</struts-config>
```

这里我们不对各个文件的内容作详细讲解，在后续章节中会逐步展开对相关关键内容的详细阐述。

2.2　Struts 基本原理与核心组件

2.2.1　Struts 基本原理

Struts 的运行过程中各个组件是通过配置文件管理并协调组织起来的。我们将通过图 2-5，初步认识 Struts 运行过程中的核心组件以及组件间的协调关系。

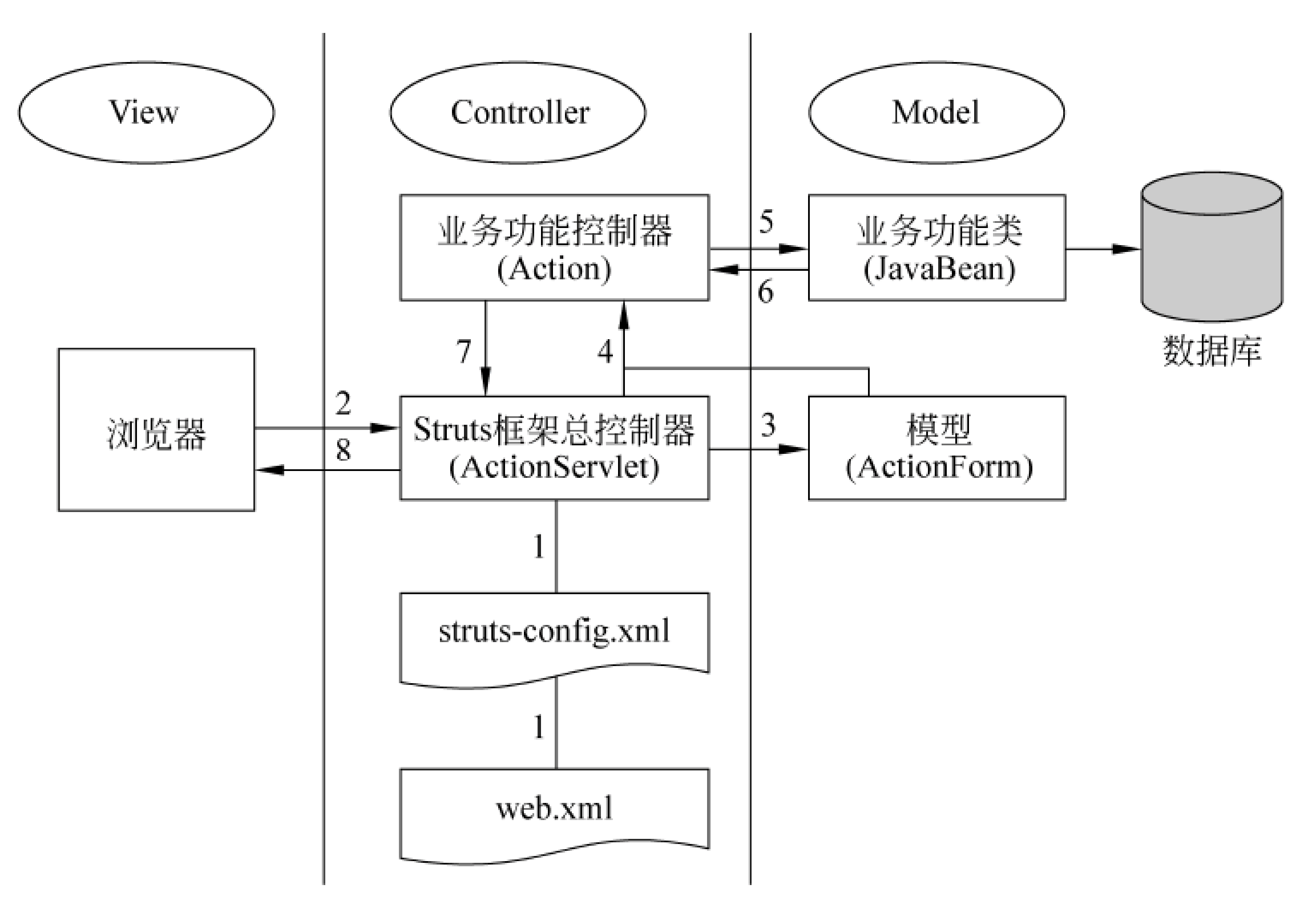


图 2-5　Struts 运行原理

当一个 Struts 应用程序开始运行时，首先把 Web 应用的配置文件 web. xml 读入内存，再通过 web. xml 读取指定的 Struts 配置文件 struts-config. xml。

在 web. xml 文件中，配置了 Struts 框架的总控制器 ActionServlet。系统通过解析 Web 的配置文件和 Struts 的配置文件，开始了 Struts 应用程序的运行过程。具体过程如下：

1. 初始化。Web 应用程序启动运行时，读取配置文件 web. xml 和 struts-config. xml，根据文件内容对 Web 应用进行初始化。

2. HTTP 请求。用户在对应视图上发送出的 HTTP 请求都被提交给 Struts 框架的总控制器 ActionServlet。

3. 填充 FormBean。ActionServlet 根据在 struts-config. xml 文件中预定义的信息，把 HTTP 请求中的表单数据填充到对应的 ActionForm 类的子类实例中。

4. 将请求转发到具体 Action 进行处理。ActionServlet 根据在 struts-config. xml 文件中预定义的映射信息，把 HTTP 请求的操作转交给具体的 Action 类的子类实例进行处理，处理所需的数据来自配置文件中预定义的关联 ActionForm 类的子类实例。

5. 调用后台业务功能类完成业务逻辑。业务功能控制器接收了操作请求和待处理数据之后，调用后台的业务功能类来完成具体的业务逻辑。

6. 业务功能类返回处理结果。业务逻辑类把处理结果返回给具体的调用类，即 Action 类的具体子类。

7. 返回目标响应对象。Action 具体子类根据处理结果，通过 struts-config. xml 中预定义的目标对象，把目标响应对象返回给 Struts 总控制器 ActionServlet。

8. 转换 HTTP 请求对象到目标响应对象。ActionServlet 根据配置文件中的预定义映射，把与目标响应对象关联的视图呈现给用户。

至此，一个完整的 Struts 应用过程就完成了。原理中涉及的更多细节问题会在后续的学习中一一进行展开。我们将在稍后的学习中再结合应用实例来理解本节讲述的 Struts 运行机制和原理。

2.2.2 struts-config. xml 配置文件

在 2.1 节我们已经看到了一个空的 Struts 配置文件，这里我们先看一个已经执行了登录功能的 Struts 环境中的配置文件，如代码清单 2-4 所示。

代码清单 2-4 登录功能实现后的 Struts 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/
struts-config_1_1.dtd">
<struts-config>
    <data-sources />
    <form-beans>
        <form-bean name="userForm" type="com.wyl.struts.form.UserForm" />
    </form-beans>
```

```
        <global-exceptions />
        <global-forwards />
        <action-mappings>
            <action attribute="userForm" name="userForm" parameter="op"
                path="/userAction" scope="request"
                type="com.wyl.struts.action.UserAction">
                <forward name="index" path="/index.jsp" />
                <forward name="login" path="/login.jsp" />
            </action>
        </action-mappings>
        <message-resources parameter="com.wyl.struts.ApplicationResources" />
</struts-config>
```

在上述代码文件中，根元素是<struts-config>，它包含两个主要元素<form-beans>和<action-mappings>，分别描述了系统中 ActionForm 对象和 Action 对象。此外一般的 struts-config. xml 还包括<global-forwards>、<global-exceptions>、<data-sources>、<message-resources>、<controller>和<plug-in>等元素。关于各个元素的作用简单介绍如表 2-1 所示。

表 2-1 struts-config. xml 的主要元素及作用

元　素　名	元素的作用
<struts-config>	根元素
<form-beans>	描述一组 ActionForm 对象
<action-mappings>	描述一组 Action 对象
<global-forwards>	定义在整个应用程序内可见的全局转发
<data-sources>	定义数据源
<global-exceptions>	定义全局异常
<message-resources>	用于配置消息资源包
<controller>	用于配置控制类
<plug-in>	用于定义添加至 Struts 应用中的插件

下面我们将详细介绍本章的应用中用到的两个元素：<form-beans>和<action-mappings>。

(1) <form-beans>元素允许配置多个 ActionForm 类。它可包含零个或多个<form-bean>子元素。<form-bean>元素的主要属性如表 2-2 所示。

表 2-2 <form-bean>元素的主要属性

属　性	描　　述
className	可选。指定和<form-bean>元素对应的配置类，默认值为 org. apache. struts. config. FormBeanConfig，自定义的配置类必须扩展该类
name	指定该 ActionForm 的唯一标识名，Struts 框架用该标识名来引用这个 form-bean。该属性必须指定
type	指定 ActionForm 的完整类路径和类名，该属性必须指定

(2) <action-mappings>元素帮助进行框架内部的流程控制，可将请求 URI 映射到 Action 类，将 Action 对象与 ActionForm 对象关联。该元素内可以定义多个<action>子元素。<action>元素所描述的是特定的请求路径和一个相应的 Action 类之间的映射关系。Struts 控制器会将请求中的 URI 路径与诸多<action>元素中的 path 属性相匹配，以选出一个特定的映射。<action>元素的属性如表 2-3 所示。

表 2-3　<action>元素的主要属性

属　性	描　　述
attribute	设置和 Action 类关联的 form-bean 在 request/session 内的属性 key，通过 request/session 的 getAttribute(attribute)方法返回该 form-bean 的实例
className	和<action>元素对应的配置元素。默认为 org. apathe. struts. action. ActionMapping
forward	指定局部转发项
include	指定包含的 URL 路径
input	指定表单输入页面的 URL 路径。当表单验证失败时将请求转发到该 URL
name	指定同该 Action 类关联的 form-bean 的引用标识名
path	指定 Action 类的访问路径，即该 Action 类在引用中的标识名，以“/”开头
parameter	指定允许调用该 Action 类的参数。在 Action 类的 execute()方法中调用 ActionMapping 对象的 getParameter()方法来读取
roles	指定允许调用该 Action 类的安全角色，多个角色之间用逗号隔开
scope	指定同该 Action 类关联的 form-bean 的存在范围。可选 request 和 session，默认是 session
type	指定该 Action 类的完整类路径和类名
unknown	若该属性为 true，则可以处理用户发出的无效 URL。默认为 false
validate	指定是否调用 form-bean 的 validate()方法验证表单的数据。默认为 true

<action>元素中可包含零个或多个<forward>子元素。<forward>子元素定义了转发动作。它的主要属性如表 2-4 所示。

表 2-4　<forward>元素的主要属性

属性	描　　述	属性	描　　述
name	指定转向标识名	redirect	指定是转发还是重定向
path	指定转向标识名所对应的转向页面		

2.2.3　Struts 控制器组件

控制器组件主要包括 ActionServlet(框架总控制器)和自定义 Action(业务控制器)。ActionServlet 类是 Struts 框架的内置核心控制器组件，它继承了 javax. servlet.

http. HttpServlet 类。Struts 的启动通常从加载 ActionServlet 开始。Web 容器会在首次启动 Struts 应用的第一个请求到达时加载 ActionServlet。ActionServlet 第一次被加载后，执行 Struts 框架的初始化工作，包括配置文件的读取等。

Action 类是 Struts 的心脏，也是客户请求和业务操作间的桥梁，每个 Action 类通常被设计为代替客户完成某种控制操作。在 Struts 应用中，具体的 Action 类需要扩展 org. apache. struts. action. Action 类，以提供 execute()方法的实现。execute()方法有 4 个参数：ActionMapping 对象、ActionForm 对象、HttpServletRequest 对象和 HttpServletResponse 对象。ActionForm 对象中封装了表单数据，因此 Action 类可通过 getter()方法从该对象中获得表单数据，然后调用模型组件处理这些数据。Action 类又通过 ActionMapping 对象的 findForward()方法获得一个 ActionForward 对象，然后把处理结果转发到 ActionForward 对象所指的目标。

ActionMapping 存储了与特定请求对应的特定 Action 的相关信息，例如输入页面、转发页面等。ActionServlet 将 ActionMapping 传送到 Action 类的 execute()方法，然后 Action 将调用 ActionMapping 的 findForward()方法，此方法返回一个指定名称 ActionForward，这样 Action 就完成了本地转发。若没有找到具体的 ActionForward，就返回一个 null。

ActionForward 对象代表一个 Web 资源的逻辑抽象表示形式。这里的 Web 资源通常是 JSP 页面或 Java Servlet。ActionForward 是该资源的包装类，所以应用程序和实际资源之间并没有多少关联。实际的 Web 资源只在配置文件 struts-config. xml 中指定(如 forward 标签的 name、path 属性)，而并非在程序代码中写入。

要从一个 Action 类返回一个 ActionForward 对象，更常见的做法是调用 ActionMapping 类的 findForward()方法找出配置文件中预先配置的一个 ActionForward 实例。

关于这几个控制类和配置文件之间的关系，请在作应用联系的过程中结合原理仔细品味，以加深理解。

2.2.4　Struts 的 FormBean

1. ActionForm

ActonForm 是 Struts 框架中的另外一个重要组件，它接收并存放来自表单的数据。ActionForm 本质上是一种 Java Bean，是专门用来传递表单数据的，它包括用于表单数据验证的 validate()方法和用于数据复位的 reset()方法。

Struts 框架利用 ActionForm 对象来临时存放视图页面中的表单数据。ActionForm 有请求(request)和会话(session)两种作用域。如果 ActionForm 的作用域设定为 request，ActionForm 实例将保存在 request 对象中，像其他保存在 request 对象中的属性一样，仅当在当前的请求响应范围内有效。如果 ActionForm 的作用域设定为 session，那么 ActionForm 实例将被保存在 session 对象中，同一个 ActionForm 实例在整个 HTTP 会话中有效。

要使用 ActionForm，既要编写 Java 类，也要在 Struts 的配置文件 struts-config. xml

的<form-beans>标签中进行配置。具体的配置方式请参考 2.2.2 节 struts-config.xml 配置文件中的内容。

2. 动态 ActionForm

本章案例中使用的是 ActionForm，另外还有一种动态 ActionForm，我们将在本书的案例“学生成绩与课程管理系统”中主要使用动态 FormBean。

动态 ActionForm 支持在 Struts 配置文件 struts-config.xml 中完成 ActionForm 的全部配置，无须编写任何额外的 Java 代码。对于 DynaActionForm 来说，在 Struts 配置文件中，<form-bean>标签的 type 属性设置为 DynaActionForm 或其子类，并在<form-bean>中包含一系列<form-property>子元素来动态标记动态表单的属性。

同 ActionForm 一样，Action 类和 JSP 都可以访问动态 ActionForm。访问 ActionForm 和访问普通 ActionForm 的最大区别在于对属性的访问方式不一致。在标准 ActionForm 中，针对每个属性都提供了 get() 和 set() 方法来读取和设置属性。而 DynaActionForm 把所有的属性都保存在一个 Map 对象中，因此访问 DynaActionForm 中的属性与访问 Map 对象中的方法类似，方法如下：

```
public Object get(String name)
public void set(String name,Object value)
```

对于<form-bean>的定义如代码清单 2-5 所示。

代码清单 2-5 <form-bean>的定义

```
<form-bean name="studentForm"
        type="org.apache.struts.validator.DynaValidatorForm">
<form-property name="id" type="java.lang.String"></form-property>
<form-property name="name" type="java.lang.String"></form-property>
<form-property name="password" type="java.lang.String"></form- property>
<form-property name="jiguan" type="java.lang.String"></form-property>
<form-property name="department" ype="java.lang.String"></form-property>
<form-property name="sex" type="java.lang.String"></form-property>
<form- property name="mark" type="java.lang.String"></form-property>
<form-property name="tel" type="java.lang.String"></form-property>
<form-property name="phone" type="java.lang.String"></form-property>
<form-property name="email" type="java.lang.String"></form-property>
</form-bean>
```

在 Action 中访问表单数据的方法为：

```
DynaActionForm stuForm= (DynaValidatorForm)form;
String name= (String)stuForm.get("name");
String password= (String)stuForm.get("password");
```

> **提示：** DynaValidatorForm 是 DynaActionForm 的子类。所有动态表单的<form-bean>配置中的 type 属性值都必须是 DynaActionForm 或其子类。

2.3　开发基于 Struts 的应用

2.3.1　需求说明

关于本原型的需求请参考 1.3.1 节。唯一需要说明的是，本章的原型系统需要采用 Struts 框架进行开发。

采用 Struts 框架开发需要遵守的设计流程图如图 2-6 所示。

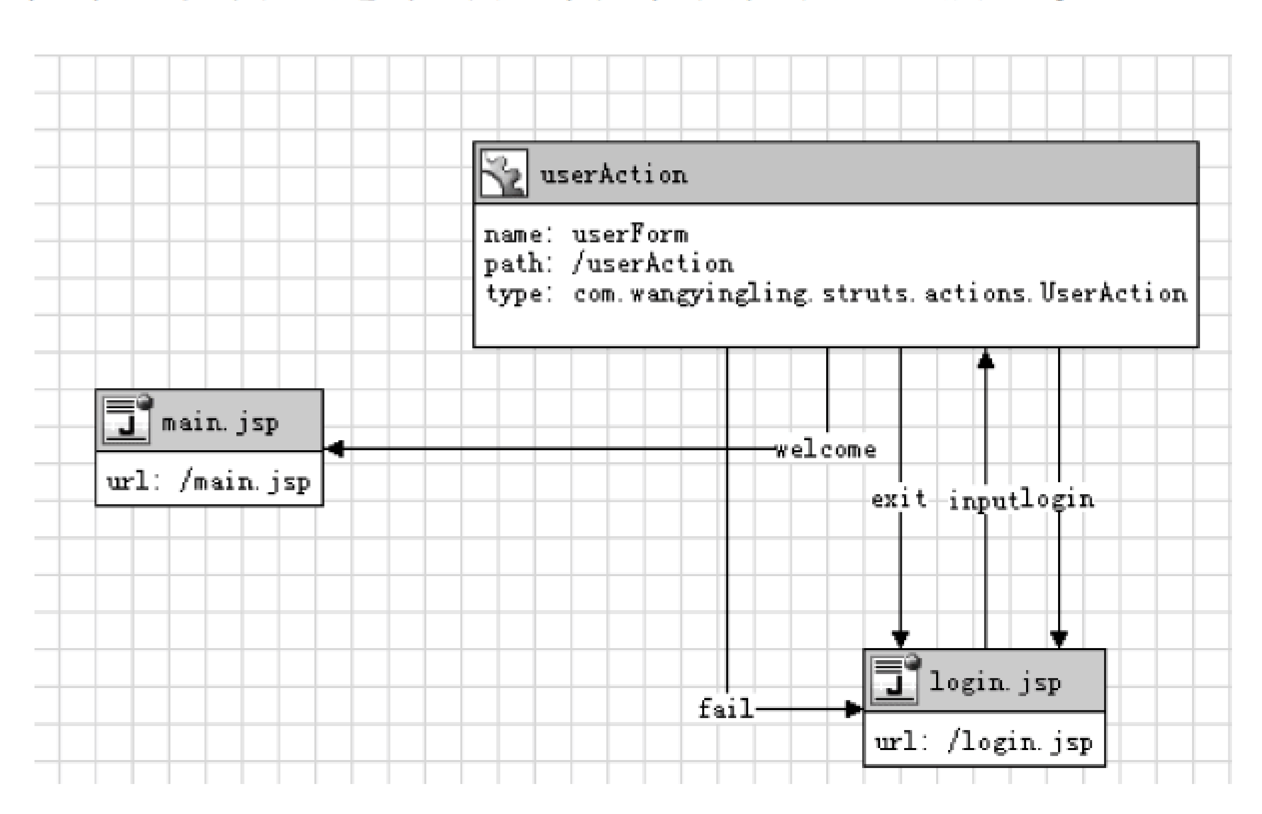


图 2-6　Struts 框架开发的原型系统设计图

2.3.2　开发基于 Struts 的用户登录功能

现在我们对该功能进行三层架构的基本分析，可以发现各层的组件如下：

（1）视图层组件：login.jsp、main.jsp、register.jsp。

（2）控制层组件：LoginAction.java。

（3）模型层组件：UserHandlerBean.java、LoginForm.java、DbBase.java。

下面我们将分层分步骤地实现该登录功能。实现步骤如下：

1. 工程准备

打开 2.1 节准备好的 Struts 工程 strutsDemo。

2. 模型层的实现

（1）创建新类 DbBase.java，位于包 com.wangyingling.struts.model 中。编辑 DbBase.java 代码，使其代码如代码清单 1-6 所示。

（2）创建新类 UserHandlerBean.java，位于包 com.wangyingling.struts.model 中。编辑 UserHandlerBean.java，使其代码如代码清单 1-5 所示（15 页）。

（3）创建 LoginForm.java。因为 MyEclipse 开发工具提供了在创建 ActionForm 子类的同时创建对应的视图页面的功能，所以我们在创建 LoginForm 的同时可以在向导中选择创建与之对应的视图页面。

- 右击选中 src 文件夹，选择菜单 new→Other。

• 在弹出的窗口中选择 MyEclipse→Web-Struts→Struts 1.2→Struts 1.2 Form，如图 2-7 所示。

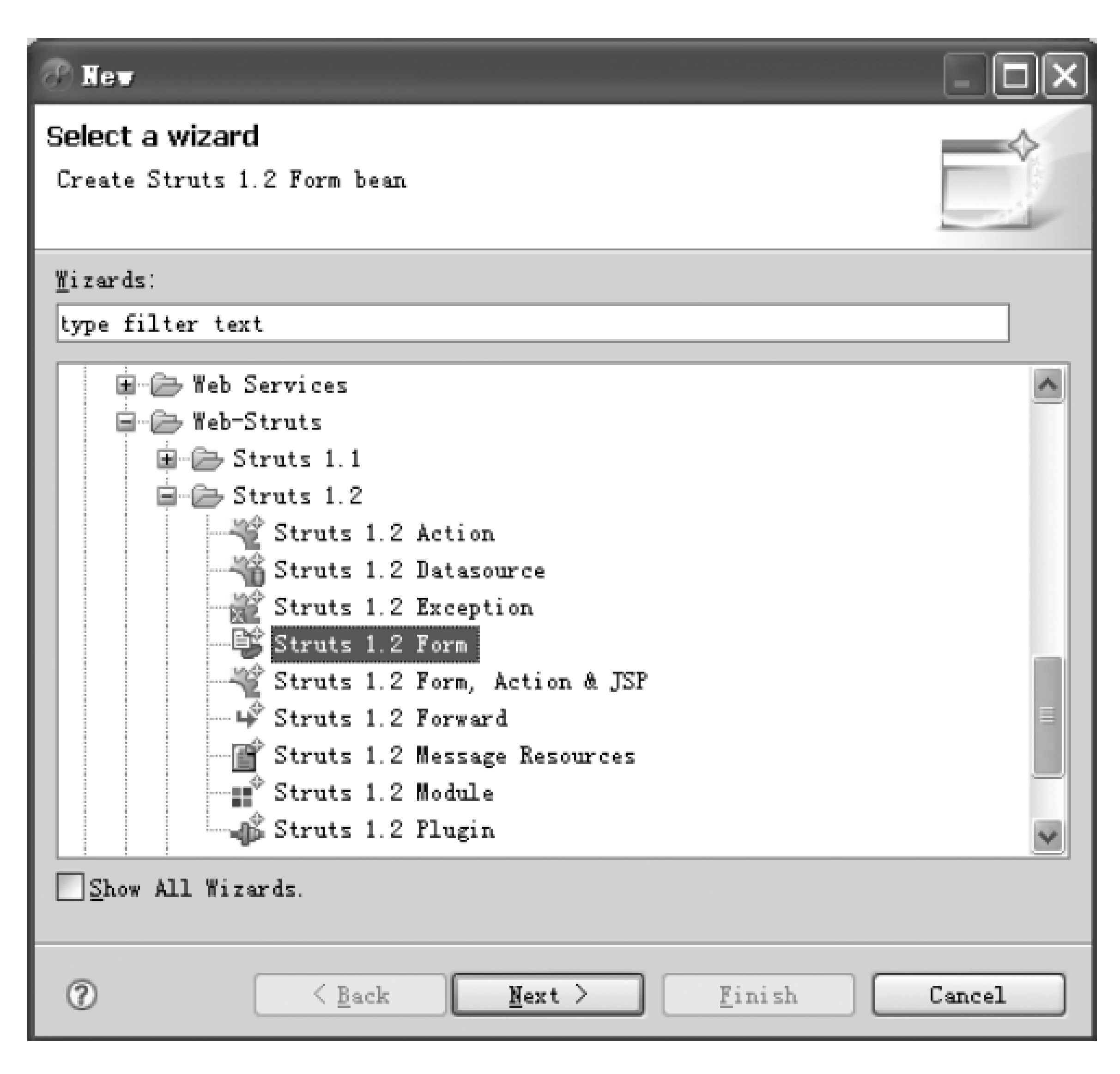


图 2-7 新建 ActionForm 之 LoginForm

• 在弹出的窗口中进行 LoginForm 的参数设置，如图 2-8 和图 2-9 所示。

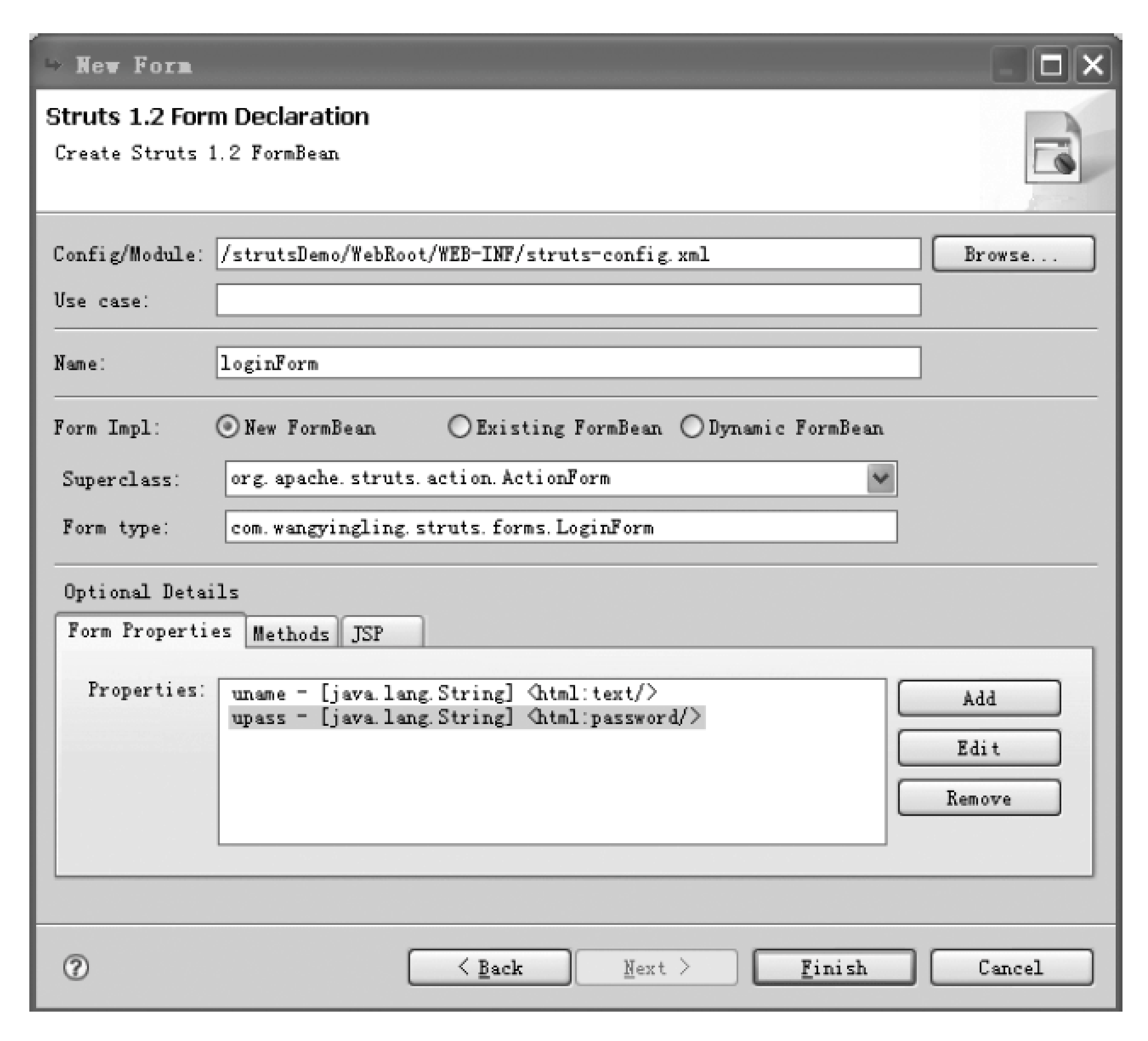


图 2-8 LoginForm 的设置

• 查看结果。在向导中的配置已经生成对应的代码存在于 struts-config.xml 文件中，并且已经生成了一个视图页面 login.jsp。现在打开 struts-config.xml 文件，

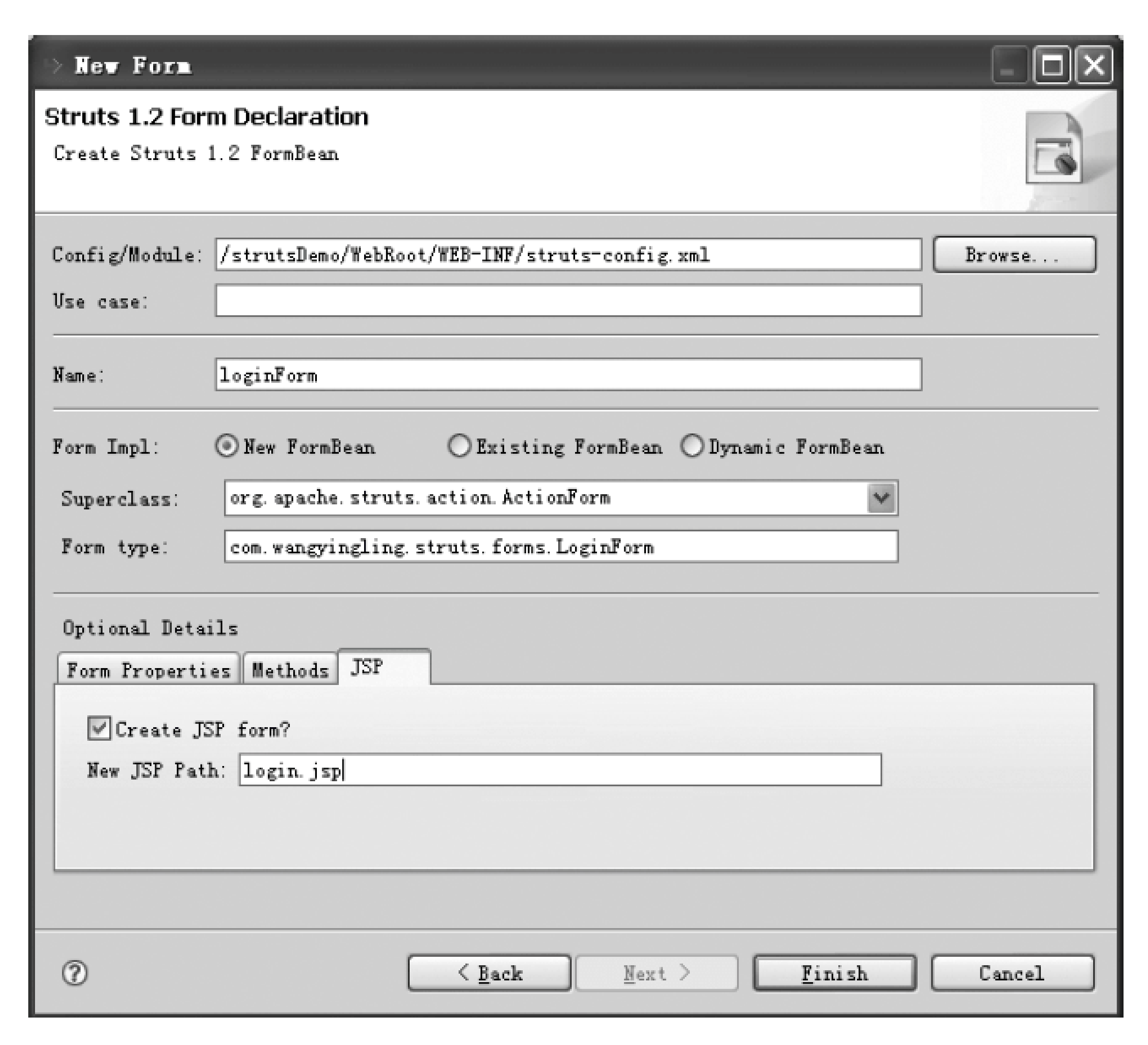


图 2-9　新建与 LoginForm 对应的视图 login.jsp

可以看到已经生成了关于 LoginForm 类的配置信息，如代码清单 2-6 和代码清单 2-7 所示。

代码清单 2-6　LoginForm 对应的配置代码

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.2//EN"
"http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
  <data-sources />
  <form-beans>
    <form-bean name="loginForm" type="com.wangyingling.struts.forms.LoginForm" />
  </form-beans>

  <global-exceptions />
  <global-forwards />
  <action-mappings />
  <message-resources parameter="com.wangyingling.struts.ApplicationResources" />
</struts-config>
```

代码清单 2-7　login.jsp

```
<%@ page language="java" pageEncoding="GBK"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<html>
```

```
    <head>
        <title>JSP for LoginForm form</title>
    </head>
    <body>
        <html:form action="/[ACTION-PATH]">
            用户名 :<html:text property="uname" />
            <html:errors property="uname" />
            <br />
            密码 :<html:password property="upass" />
            <html:errors property="upass" />
            <br />
            <html:submit />
            <html:cancel />
        </html:form>
        如果尚未注册,
        <html:link target="_self" href="register.jsp">请单击这里</html:link>
    </body>
</html>
```

3. 控制层组件的实现

- 右击选中 src 文件夹,选择菜单 new→Other。
- 在弹出的窗口中选择 MyEclipse→WebStruts→Struts 1.2→Struts 1.2 Action。如图 2-10 和图 2-11 所示,输入 LoginAction 的配置信息。

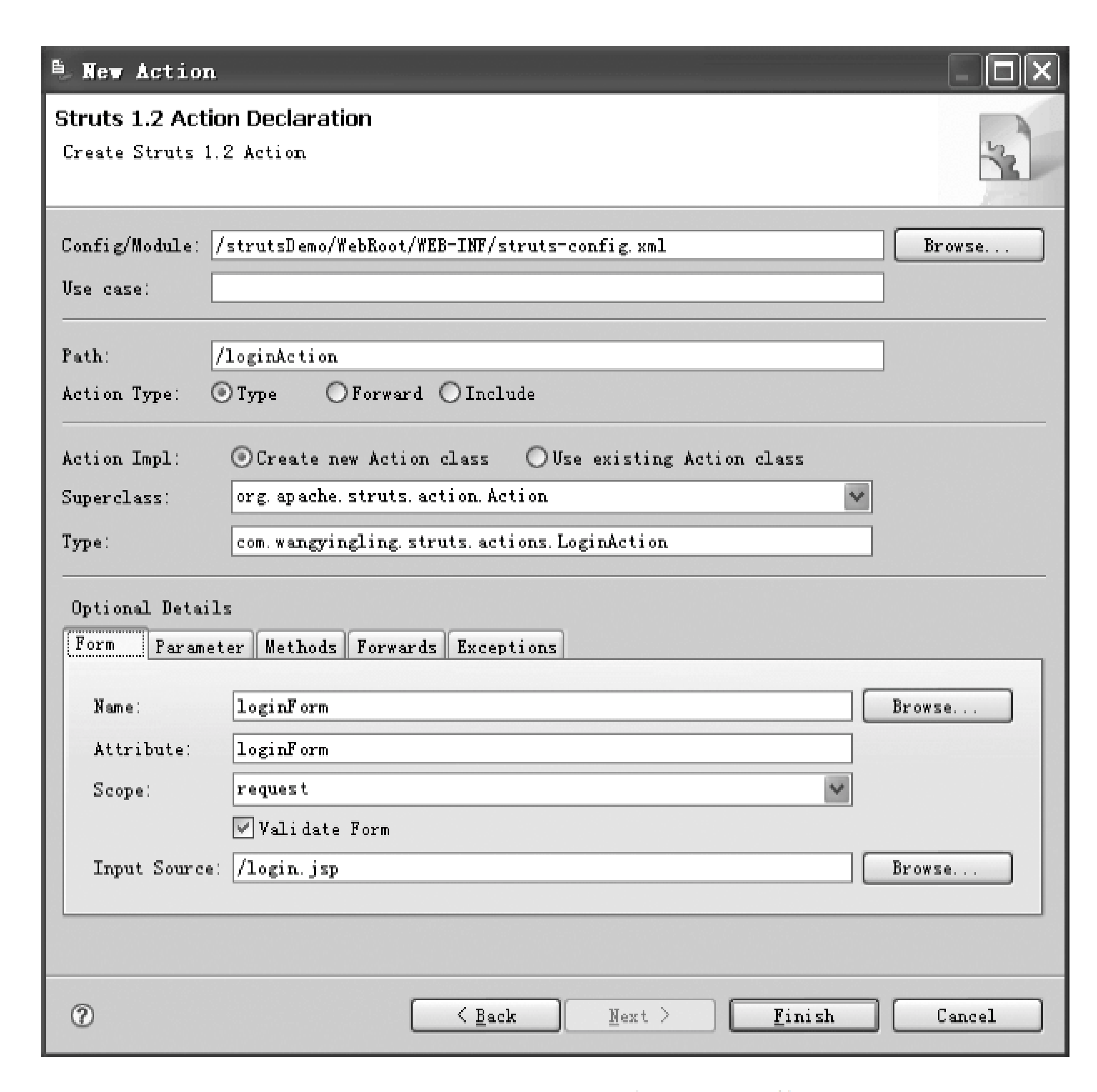


图 2-10 LoginAction 类的配置信息

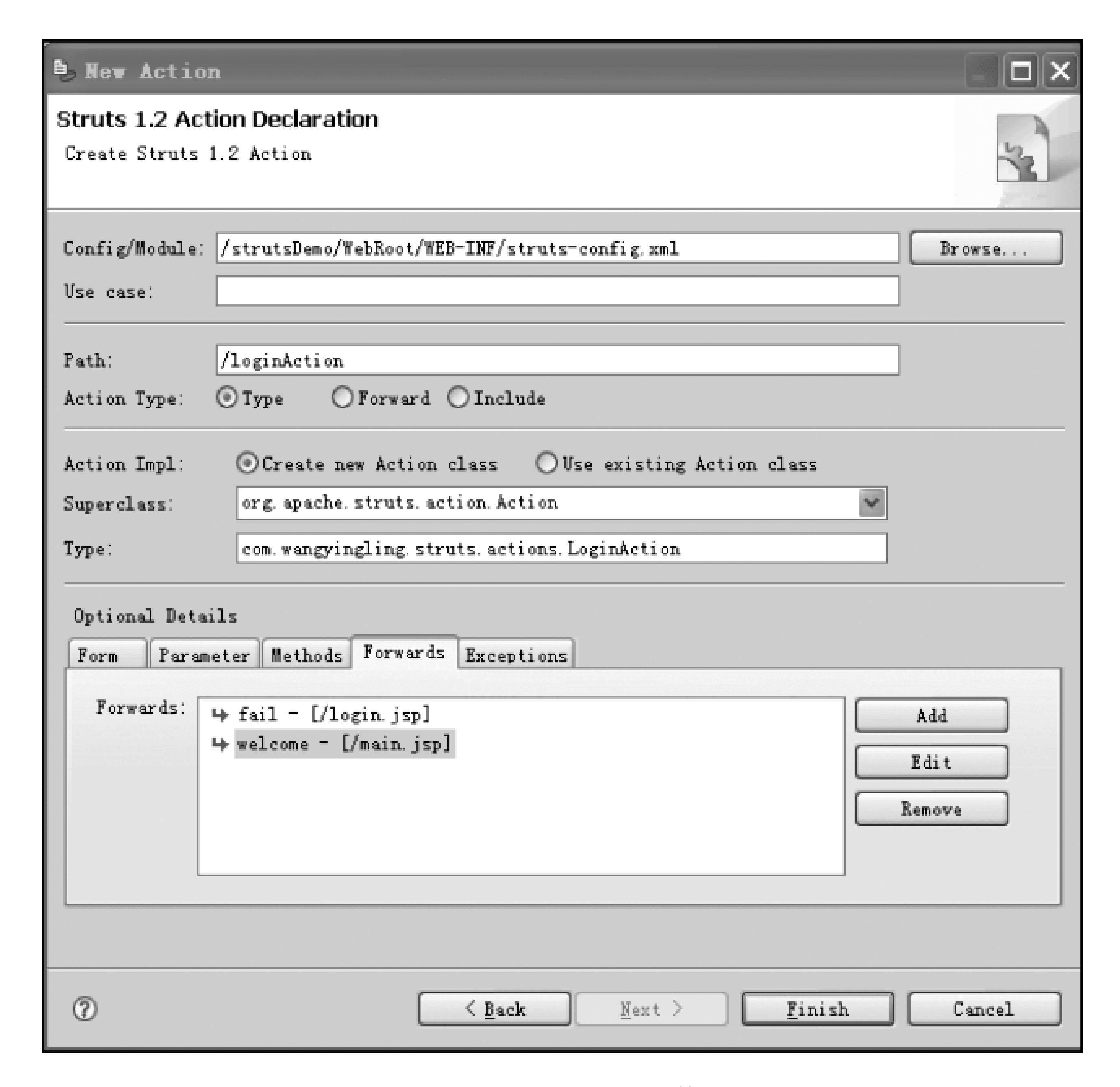


图 2-11　LoginAction 的配置信息之 forward

- 单击 Finish 按钮完成 UserAction 的创建。现在可以看到 struts-config.xml 文件中有了 LoginAction 对应的配置代码，如代码清单 2-8 所示。

代码清单 2-8　LoginAction 对应的配置信息

```
< ?xml version= "1.0" encoding= "UTF- 8"?>
< !DOCTYPE struts- config PUBLIC "- //Apache Software Foundation//DTD
Struts Configuration 1.2//EN"
"http://struts.apache.org/dtds/struts- config_1_2.dtd">

< struts- config>
  < data- sources />
  < form- beans>
    < form- bean name= "loginForm" type= "com.wangyingling.struts.forms.LoginForm" />
  < /form- beans>

  < global- exceptions />
  < global- forwards />
  < action- mappings>
    < action
      attribute= "loginForm"
      input= "/login.jsp"
      name= "loginForm"
      path= "/loginAction"
```

```
          scope="request"
          type="com.wangyingling.struts.actions.LoginAction">
          <forward name="welcome" path="/main.jsp" />
          <forward name="fail" path="/login.jsp" />
        </action>
    </action-mappings>
    <message-resources parameter="com.wangyingling.struts.ApplicationResources" />
</struts-config>
```

- 编辑 LoginAction 类完成登录控制器逻辑，编辑后的代码如代码清单 2-9 所示。

代码清单 2-9 LoginAction. java

```
package com.wangyingling.struts.actions;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import com.wangyingling.struts.forms.LoginForm;
import com.wangyingling.struts.model.UserHandlerBean;
public class LoginAction extends Action {
    /**
     * Method execute
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @return ActionForward
     */
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response){
        LoginForm loginForm= (LoginForm) form;
        ActionErrors errors=new ActionErrors();
        ActionForward forward=new ActionForward();
        try {
            //获得请求参数
            String username=loginForm.getUname();
            //如果会话存在，则使其失效。
            HttpSession session=request.getSession(false);
            if (session !=null){
                session.invalidate();
```

```
                }
                //为当前用户创建一个会话
                session= request.getSession(true);
                //执行登录操作
                boolean isValid= valid(request, loginForm);
                if (isValid){
                    session.setAttribute("userName", username);
                } else {
                    errors.add(ActionErrors.GLOBAL_MESSAGE, new ActionMessage(
                            "login.message.failed"));
                }
            } catch (Exception e){
                errors.add(ActionErrors.GLOBAL_MESSAGE, new ActionMessage(
                        "login.message.failed"));
            }
            //If a message is required, save the specified key(s)
            //into the request for use by the< struts:errors> tag.
            if (!errors.isEmpty()){
                saveErrors(request, errors);
                request.setAttribute("loginFormBean", loginForm);
                forward= mapping.findForward("fail");
            } else {
                forward= mapping.findForward("welcome");
            }
            //Finish with
            return (forward);
        }
        private boolean valid(HttpServletRequest request, LoginForm loginForm){
            UserHandlerBean uhb= new UserHandlerBean();
            Boolean isValid= uhb.valid(loginForm.getUname(), loginForm.getUpass());
            return isValid;
        }
    }
```

4. 视图层组件的实现

1）修改 login. jsp

在代码清单 2-7 中，将代码<html:form action="/[ACTION_PATH]">修改为<html:form action="/loginAction">，以便该视图上的请求会被转发到对应的 Action 子类 LoginAction 类进行处理。

2）创建 main. jsp，编辑使其内容如代码清单 2-10 所示

代码清单 2-10　main. jsp

```
<%@page language="java" import="java.util.* " pageEncoding="GB18030"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>
    <head>
        <title>欢迎光临 struts示例主页面</title>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
    </head>
    <body>
        <div align="center">
            <font size="4"><strong><font color="#008000">
                        ${sessionScope.userName }</font>你好!欢迎光临!<br><br>
                    <form action="exitAction.do" method="post">
                        <input type="hidden" value="logout" name="opType">
                        <input type="submit" name="submit" value="退出">
                    </form><br></strong></font>
        </div>
    </body>
</html>
```

5. 发布 Web 项目

- 选中项目 strutsDemo。
- 单击工具栏上的发布图标：。
- 在弹出的对话框中单击 Add 按钮。
- 在弹出的对话框中 Server 下拉框中选择 Tomcat 5.x 服务器，然后确定。

6. 测试运行

- 在 Servers 视图中，选择 Tomcat5.x 服务器，并单击右上的 debug 图标：。
- 当控制视图中出现 Server started in * minites 时，服务器运行成功。
- 打开浏览器，输入地址：http://localhost:8080/strutsDemo/login.jsp 即可打开登录页面如图 2-12。

输入适当的数据进行登录验证即可。



图 2-12 登录页面

2.3.3 结合案例回顾 Struts 原理

1. Struts 应用中的操作回顾

现在我们对 Struts 的原理和 Struts 应用的步骤都已经有了一定的认识，那么我们现在就结合我们的操作步骤，再来理解操作步骤的各项设置和 Struts 原理分别有什么关

系，从而加深理解。

1）视图层的操作

我们在每一次的操作配置完成之后，通过观察 struts-config. xml 文件可以发现，对应的配置选项的编辑，都生成了对应的配置代码。具体的对应关系如表 2-5。

表 2-5　FormBean 向导中的设置项与 struts-config. xml 文件的关系

New Form 操作界面项/输入框	struts-config. xml 文件的标签及属性
Name 文本框	<form-bean>的 name 属性
formType 文本框	<form-bean>的 type 属性
FormProperties 选项卡	生成的 FormBean 类中的属性及其类型
jsp 选项卡	生成与当前 FormBean 对应的 jsp 页面，并根据当前 FormBean 的属性生成页面中的 form 表单及其项

2）控制层的操作

同视图层的操作一样，我们在新建 Action 子类，并进行向导中相应项的设置时，不仅仅是要新建一个 Action 类，同时也在配置与之对应的 struts-config. xml 文件中的 action 标签及其属性。其对应关系见表 2-6。

表 2-6　Action 类向导中的设置项与 struts-config. xml 文件的关系

操作界面项/输入框	struts-config. xml 文件的标签及属性
Path 文本框	<action>标签的 path 属性，必须以“/”开头
Type 文本框	<action>标签的 type 属性，其值为 Action 子类的全路径名
Form 选项卡中的 Name 文本框	<action>标签的 name 属性，其值为与 Action 子类对应的 FormBean 的名字，即 action 执行时的数据来源
Form 选项卡中的 Scope 文本框	<action>标签的 scope 属性，有 page、request 和 session 三个值可选，表示了 FormBean 中存放的数据的有效期范围
Form 选项卡中的 InputSource 文本框	<action>标签的 input 属性，当数据验证失败时要跳转至的视图
Forward 选项卡	<action>标签的子标签<forward>
Forward 选项卡中的 Name	<forward>标签的 name 属性，表示跳转路径的映射名
Forward 选项卡中的 Path	<forward>标签的 path 属性，表示实际的跳转路径

2．用 Struts 原理图解释登录程序

现在我们把已经完成的登录程序运行起来，并结合 Struts 原理，重新理解其运行过程：

（1）程序启动，Web 容器加载 web. xml 和 struts-config. xml 进行初始化。

（2）用户在登录页面输入用户名和密码，其值分别存储在变量 uname 和 upass 中。然后单击“登录”按钮，请求信息被发送给控制器 ActionServlet 类。

（3）根据配置文件 struts-config. xml 文件中的预配置信息，ActionServlet 把请求数

据存放在配置文件指定的 loginForm 对象中，并把操作请求转发给配置文件指定的 loginAction 对象。

(4) loginAction 对象接收操作请求，并从配置文件中指定的 loginForm 对象中读取请求数据。

(5) loginAction 对象调用 Model 层的相应接口来处理请求数据，执行请求操作。

(6) 模型层返回请求的处理结果，给 loginAction 对象。

(7) loginAction 对象根据模型层的处理结果，把对应的视图映射(fail 或者 welcome)封装成 ActionForward 对象返回给 ActionServlet。

(8) ActionServlet 根据配置文件的配置信息，把对应的视图呈现给用户。

2.4 实验与能力拓展

1. 请根据你所了解的登录判断逻辑，完善 2.3 节的应用程序中模型层关于登录判断接口的逻辑。

2. 假如我们的应用程序，因为开发过程没有遵守开发规范，把视图层的 LoginForm.java 类的名字写错了，现在需要把 LoginForm.java 重新命名为 UserForm.java，那么我们需要在不重新创建文件的前提下，修改程序中的哪些内容，才可以使程序正常运行？

3. 如果我们的控制层 LoginAction.java 的访问路径要修改为“login.do”，那么我们需要对原有的应用程序做哪些修改？

4. 请根据登录的 Struts 实现过程，实现第 1 章要求的注册功能的 Struts 开发实现。

5. 请根据登录的 Struts 实现过程，实现第 1 章要求的退出功能的 Struts 开发实现。

第 3 章　使用 DispatchAction 优化控制层

本章导读

我们在前面章节中提到过 Struts 架构主要是对 MVC 模式的控制层和视图层规范进行了技术实现，在第 2 章我们已经初步认识了开发 Struts 应用的过程和 Struts 运行机制。本章我们将从对 Struts 控制层的认识开始，并基于项目的实际应用掌握一种 Struts 控制层开发的优化方法，即 DispatchAction 类的应用。

通过本章的学习，读者能够掌握控制层中各个类之间的协作关系，以及如何使用 DispatchAction 类来优化控制层的开发。

工作任务

使用 DispatchAction 优化原型系统，完成登录、注册、退出功能。

3.1　默认的 Action 类

3.1.1　默认的 Action

这里我们所说的默认 Action 类指的是类 org. apache. struts. action. Action. java。具体的 Action 类继承该类，并且具体 Action 类的功能一般都在 execute()方法中完成。在第 2 章的案例中我们采用的就是这种 Action 类的子类来完成控制器的功能。

3.1.2　解读 Action 类的 execute()方法

继承 org. apache. struts. action. Action. java 的具体子类的名称推荐使用 Action 结尾。具体子类必须重写 execute()方法。

execute()方法的签名如下：

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response)
```

在 execute()方法中需要通过编码完成下列工作：

- 接收请求参数和请求数据。ActionForm 对象 form 中传递进来的是请求表单中的参数，request 和 response 中分别存放请求和响应信息，其功用和 servlet 中接收的请求和响应一致。
- 编写业务逻辑，实现业务功能。通常可以通过调用 Model 层的业务接口实现。
- 根据业务功能的反馈结果，决定程序流程方向。输入参数 mapping 中存放的是在 struts-config.xml 文件中定义的映射关系，如跳转路径名(即 forward 标签中的 name 属性)和实际路径(forward 标签中的 path 属性)间的映射关系。根据处理结果，把程序流向(如视图跳转目标)封装成 ActionForward 对象的形式返回给 ActionServlet 类。

> **提示：** 在 loginAction 的 execute()方法中我们实现了数据验证，一般情况下，本着 MVC 分离的原则，也就是视图级的验证工作放在视图端来完成，可以由 ActionForm 来完成，也可以在前端通过 JavaScript 脚本来完成。

当用户请求由 Struts 框架的总控制器 ActionServlet 分发给具体的 Action 子类时，Action 子类对象会执行 execute()方法。这种默认 Action 类的子类只有一个 execute()方法，即同一个 Action 具体子类只能处理一种业务逻辑(或者通过在 execute()方法内编写复杂的选择分支也可以达到处理多种业务逻辑的目的)。

3.2 DispatchAction

3.2.1 使用 DispatchAction 的必要性

在第 2 章中，我们实现了登录、注册和退出功能，解决方案是每个视图页面对应一个 Action 具体子类，每个 Action 具体子类对应一个 ActionForm 具体子类。

随着开发的进行我们的原型系统要进行扩展会添加更多的视图页面和功能进来。项目功能扩展完善之后，肯定会有一个页面多个功能按钮，或者一个页面一个功能按钮的情形出现，按照第 2 章的解决方案，有多少个页面就会有多少个 Action 具体子类，甚至更多。另外一种解决方案是把操作请求的判断逻辑写在 Action 具体子类的 execute()方法代码内部。

无论哪种解决方案，都会给我们的项目维护带来困难。一方面，会使我们的项目规模大到难以控制；另一方面，我们的项目会出现冗余代码，使 Action 具体子类内的逻辑复杂，不易阅读维护。那么我们有什么更好的解决方案吗？

Struts 框架中提供的 DispatchAction 就可以很好地解决这种问题。

3.2.2 DispatchAction 的使用

通常，一个 Action 中只能完成一种业务操作，通过扩展 org.apache.struts.actions.

DispatchAction 实现的 Action 类可以完成一组相关的几种业务操作。DispatchAction 类称为可自动分发的 Action，它的作用就是实现按业务实体划分类。

扩展自 DispatchAction 的 Action 类中不必定义 execute()方法，而是创建一些实现实际业务操作的方法，如 doReg()、doLogin()等。但是这些业务方法的传入参数和返回值要和普通 Action 的 execute()方法相同，并声明可能抛出同样的异常。

而且 DispatchAction 类内部的业务处理方法的名字是取决于传递过来的 form 表单的参数名的。

我们要使用 DispatchAction 类进行系统的优化，需要参考以下步骤。

(1) 对业务实体进行识别，然后把对同一个业务实体的多个操作合并到一个 DispatchAction 子类中进行处理。

(2) 在 struts-config.xml 文件中的<action>标签中添加一个参数 parameter。该参数用来指定函数入口的参数名。

(3) 在视图页面的表单<form>的请求地址中传递参数“method＝test”。

(4) 在 Action 子类中修改父类为 DispatchAction，并新建参数值对应的函数。此函数的签名和 execute()方法一致，只是方法名为 method 参数的值，即和第三步中的 method 参数的值一致。

完成了上述操作之后，在运行时，就会根据参数的不同，由不同的方法自动运行不同的业务处理。

3.3　使用 DispatchAction 改进原型系统

现在我们要对第 2 章完成的原型系统进行优化，按照业务实体进行划分，登录、退出和注册功能都是对用户业务实体进行操作，所以，我们只需要一个控制 Action 类，即 DispatchAction 子类。

我们命名处理用户实体的 Action 子类为 UserAction.java，用户实体用 UserForm.java 表示。要对用户实体进行的操作分别是登录、注册和退出，所以我们分别命名这几个操作对应的方法名为 doLogin、doRegister 和 doExit。

> **提示：**在 DispatchAction 子类中，方法名称的前缀也常常遵循一定的惯例。
> (1) 转到编辑页面的方法常命名为 toEdit；
> (2) 执行编辑操作的方法常命名为 doEdit；
> (3) 以此类推，命名为 doAdd、toList、doDel 等。
> toXxx 表示转到 Xxx 页面，而 doXxx 表示执行 Xxx 操作。
> 通过这样的命名，可以使程序逻辑更清晰，减少出错的几率，便于维护。

3.3.1　使用 DispatchAction 为原型系统添加注册功能

我们参考 3.2.2 节的 DispatchAction 步骤来对原型系统进行修改，优化控制层的开发。

- 修改 struts-config. xml 文件中的 LoginAction 类对应的 action 标签。
  - 为该签添加属性 parameter=“method”。
  - 修改该 action 标签的 Type 属性的全类名为 com. wangyingling. struts. actions. UserAction。
  - Action 标签的 name 属性的值修改为 userForm。
  - Action 标签的 path 属性的值修改为/userAction。
- 修改 LoginAction 类。
  - 重命名 LoginAction 类为 UserAction。
  - 修改 UserAction 类的父类为 org. apache. struts. actions. DispatchAction。
  - 修改 execute()方法名为 doLogin。
  - 添加方法 doRegister()。

修改编辑后的 UserAction 类的源代码如代码清单 3-1 所示。

代码清单 3-1 UserAction. java

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package com.wangyingling.struts.actions;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.actions.DispatchAction;
import com.wangyingling.struts.forms.UserForm;
import com.wangyingling.struts.model.UserHandlerBean;
public class UserAction extends DispatchAction {
    public ActionForward doLogin(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response){
        UserForm loginForm= (UserForm) form;
        ActionErrors errors= new ActionErrors();
        ActionForward forward= new ActionForward();
        try {
            //获得请求参数
            String username= loginForm.getUname();
            //如果会话存在,则使其失效。
            HttpSession session= request.getSession(false);
            if (session != null){
                session.invalidate();
```

```
            }
            //为当前用户创建一个会话
            session= request.getSession(true);
            //执行登录操作
            boolean isValid= valid(request, loginForm);
            if (isValid){
                session.setAttribute("userName", username);
            } else {
                errors.add(ActionErrors.GLOBAL_MESSAGE, new ActionMessage(
                        "login.message.failed"));
            }
        } catch (Exception e){
            errors.add(ActionErrors.GLOBAL_MESSAGE, new ActionMessage(
                    "login.message.failed"));

        }
        //If a message is required, save the specified key(s)
        //into the request for use by the< struts:errors> tag.
        if (!errors.isEmpty()){
            saveErrors(request, errors);
            request.setAttribute("loginFormBean", loginForm);
            forward= mapping.findForward("fail");
        } else {
            forward= mapping.findForward("welcome");
        }
        //Finish with
        return (forward);
    }
    public ActionForward doRegister(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response){
        UserForm registerForm= (UserForm) form;           //method stub
        ActionErrors errors= new ActionErrors();
        ActionForward forward= new ActionForward();
        try {
            //判断用户名是否已存在
            boolean isExist= isExist(request, registerForm);
            if (isExist){
                errors.add(ActionErrors.GLOBAL_MESSAGE, new ActionMessage(
                        "register.message.failed"));
            } else {
                insert(request, registerForm);         //执行注册操作
            }
        } catch (Exception e){
            errors.add(ActionErrors.GLOBAL_MESSAGE, new ActionMessage(
                    "register.message.failed"));
```

```
        }
        //If a message is required, save the specified key(s)
        //into the request for use by the< struts:errors> tag.
        if (!errors.isEmpty()){
            saveErrors(request, errors);
            request.setAttribute("registerFormBean", registerForm);
            forward=mapping.findForward("fail");
        } else {
            forward=mapping.findForward("login");
        }
        //Finish with
        return (forward);
    }
    private void insert(HttpServletRequest request, UserForm registerForm){
        //调用业务层方法实现注册
        UserHandlerBean uhb=new UserHandlerBean();
        uhb.add(registerForm.getUname(), registerForm.getUpass(), registerForm
                .getEmail());
    }
    private boolean isExist(HttpServletRequest request,
            UserForm registerForm){
        //调用业务层方法判断用户名是否已经存在
        UserHandlerBean uhb=new UserHandlerBean();
        boolean isExist=uhb.isExist(registerForm.getUname());
        return isExist;
    }
    private boolean valid(HttpServletRequest request, UserForm loginForm){
        UserHandlerBean uhb=new UserHandlerBean();
        boolean isValid=uhb.valid(loginForm.getUname(), loginForm.getUpass());
        return isValid;
    }
}
```

- 修改视图页面 login.jsp、register.jsp 中的 form 表单中的 action 属性。
  - ■ login.jsp 文件中的 form 表单的 action 属性被修改为:

```
<html:form action="/userAction?method=doLogin">
```

  - ■ register.jsp 文件中的 form 表单的 action 属性被修改为:

```
<html:form action="/userAction?method=doRegister">
```

- 修改 RegisterForm 类。把 RegisterForm 类重命名为 UserForm。
- 修改 struts-config.xml 文件中的 registerForm 对应的 form-bean 标签。把该标签的 name 属性值修改为 userFrom，type 属性值修改为 com.wangyingling.struts.forms.UserForm。

- 在 userAction 的 action 中添加如下子标签：

```
<forward name="login" path="/login.jsp" />
```

- 重命名 RegisterForm.java 为 UserForm.java。

3.3.2　使用 DispatchAction 为原型系统添加退出功能

在注册功能使用 DispatchAction 实现的基础上再来修改退出功能的实现需要经过下述步骤。

- main.jsp 文件中的 form 表单的 action 属性被修改为：

```
<form action="userAction.do?method=doExit" method="post">
```

- 在 UserAction 类中添加方法 doExit()。该方法的完整代码如代码清单 3-2 所示。

代码清单 3-2　UserAction.java 中添加的 doExit()方法

```
public ActionForward doExit(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response){
        ActionForward forward=new ActionForward();

        //invalidate the original session if exists
        HttpSession session=request.getSession(false);
        String username=(String)session.getAttribute("userName");
        if (session !=null){
            session.invalidate();
        }
        forward=mapping.findForward("exit");
        //Finish with
        return (forward);
    }
```

- 在 struts-config.xml 文件中，为 userAction 的 action 标签添加子标签：

```
<forward name="exit" path="/login.jsp" />
```

完成控制层优化并删除不需要的其他内容之后，struts-config.xml 文件如代码清单 3-3 所示。

代码清单 3-3　struts-config.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.2//EN"
"http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
  <data-sources />
```

```
  <form-beans>
    <form-bean name="userForm" type="com.wangyingling.struts.forms.UserForm" />

  </form-beans>

  <global-exceptions />
  <global-forwards />
  <action-mappings>
    <action
      attribute="loginForm"
      input="/login.jsp"
      name="userForm"
      path="/userAction"
      scope="request"
      type="com.wangyingling.struts.actions.UserAction"
      parameter="method">
      <forward name="welcome" path="/main.jsp" />
      <forward name="fail" path="/login.jsp" />
      <forward name="login" path="/login.jsp" />
      <forward name="exit" path="/login.jsp" />
    </action>

  </action-mappings>

  <message-resources parameter="com.wangyingling.struts.ApplicationResources" />
</struts-config>
```

3.4 实验与能力拓展

1. 请根据本书内容，完成案例练习。

2. 请重新建一个工程 dispatchDemo，工程要求完成的内容和原型系统要求一致，但是要求从一开始就采用 DispatchAction 类来完成控制层。

3. 请尝试用 MyEclipse 的调试功能，跟踪登录、注册和退出的运行，理解采用了 DispatchAction 的原型系统的运行流程。

4. 假若我们的各个页面提交到 userAction 时，参数 method 改为 op，那么应该如何修改工程才能运行正常？

5. 使用 MyEclipse 的调试功能对原型系统进行跟踪运行，从而进一步理解参数值如何关联到 DispatchAction 的相关方法运行。修改参数值或 DispatchAction 子类中的方法名，看看会出现什么问题，并解释原因。

第 4 章　使用 Struts 标签进行页面处理和国际化

本章导读

本章关注点是页面的国际化，Struts 页面的国际化是通过 Struts 的标签来实现的。

本章的主要内容包括 Struts html 标签、struts-bean 标签、struts-logic 标签和国际化的步骤和原理。

工作任务

使用 Struts 标签和资源文件实现视图层的国际化处理。

4.1　Struts 中国际化处理

4.1.1　国际化

“国际化”是指一个应用程序在运行时能够根据客户端请求所来自的国家/地区、语言的不同而显示不同的用户界面。例如，请求来自一台中文操作系统的客户端计算机，则应用程序响应界面中的各种标签、错误提示和帮助信息均使用中文文字；如果客户端计算机采用英文操作系统，则应用程序也能识别并自动以应用界面做出响应。

可以看出，引入国际化机制的目的在于提供自适应的、更友好的用户界面，而并未改变程序的其他功能/业务逻辑。人们常用 I18N 这个词作为“国际化”的简称，其来源是英文单词 Internationalization 的首末字母 I 和 N 及它们之间的字符数 18。

Struts 框架通过使用＜bean：message＞标记，以及使用 java. util 数据包中定义的 Local 和 ResourceBundle 类来支持国际化。java. text. MessageFormat 类定义的技术可以支持消息的格式。利用此功能，开发人员无须了解这些类的细节就可进行国际化处理并设置消息格式。

在 Struts 应用中实现国际化，有两种方式。一种是在 Struts 应用程序中通过程序的方式来访问资源文件，达到国际化的目的；另外一种方式是通过在 Struts 视图中通过 Struts 标签库的相应标签访问资源文件，从而达到国际化的目的。本章的关注点是通过

Struts 标签实现 Struts 视图的国际化，其中涉及的知识会在后续章节中逐一展开，并最后给出国际化处理的流程和应用实例。

4.1.2 资源文件

1. 什么是资源文件

要用 Struts 实现国际化和本地化，最重要的资源就是支持国际化资源文件。资源文件包含使用默认语言编写的会在程序中出现的所有消息。这些消息以“键-值”的形式存储，例如：

```
error.vaildation.location=The entered location is invalid
```

资源文件是以属性文件的形式存在的，即文件名的后缀是.properties。

当对一个应用程序进行国际化处理时，所要用的各种语言版本的标签信息应该存放在不同的属性文件中，每一个这样的文件对应一种语言的版本。所有作为资源文件的属性文件合在一起称为**资源包**(Resource Bundle)。

属性文件的命名格式可以分为以下两种：

- 文件名前缀.properties。
- 文件名前缀_语言种类.properties。

文件名后缀必须为 properties，前缀则由开发者自行确定，其中的语言种类字段必须是有效的 ISO(International Standardization Organization，国际化标准组织)语言代码，ISO-639 标准定义的这些代码格式为英文小写、双字符，具体如表 4-1 所示。

表 4-1 常见语言和编码对照表

语　言	编码	语　言	编码
汉语(Chinese)	zh	德语(German)	de
英语(English)	en	日语(Japanese)	ja
法语(Franch)	fr	意大利语(Italian)	it

访问网址 http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt 可以获得 ISO-639 标准的其他语言代码。

文件名前缀中语言代码缺省的为默认属性文件。当系统中找不到与客户端请求的语言属性匹配的属性文件时，则使用默认属性文件。

例如我们要对前面的登录系统进行国际化处理，要求根据不同的语言环境显示英文和中文的用户界面，那么就需要创建英文和中文版本的资源文件，分别取名为 ApplicationResource_en.properties 和 ApplicationResourse_zh.properties。它们的内容如代码清单 4-1 和代码清单 4-2 所示。

代码清单 4-1 默认资源文件 ApplicationResource_en.properties

```
title.login=welcome to login
title.register=welcome to register
title.main=welcome to mainPage
```

```
label.username= Input your username
label.password= Input your password
label.confirm= inout your password again
label.email= input your email
label.href.register= register
item.submit= Submit
item.reset= Reset
message.success= welcome
message.failure= you fail to login
```

代码清单 4-2　中文资源文件 ApplicationResource_zh. properties

```
#一个简单的属性文件

title.login=登录页面
title.register=欢迎注册
title.main=欢迎光临主页面
label.username=姓名
label.password=密码
label.confirm password=确认密码
label.email=电子邮件
label.href.register=注册
item.submit=登录
item.reset=重置
message.success=已成功登录。欢迎光临。
message.failure=登录失败。现进入注册页,请注册你的信息。
```

可以看出,属性文件中包含的是“键-值”对形式的字符信息。属性文件在应用程序启动时被载入内存,并根据客户端请求所携带的语言/地区信息进行匹配和使用。

资源文件需要存储在类的路径下,而且它的路径要作为初始化参数传递给 ActionServlet 时,路经的格式符合完整 Java 类的标准命名规范。比如,如果资源文件存储在 WEB-INT\classes 目录中,文件名是 ApplicationResources. properties,那么需要传递的参数值是 ApplicationResources。如果文件在 WEB-INT\classes\com\test 中,那么参数值就应该是 com. test. ApplicationResources。

要想使用资源包进行 Struts 视图的国际化处理,就需要在 Struts 的配置文件 struts-config. xml 中进行配置,说明要使用的资源包。<message-resource>元素就是用来配置资源包的。例如对上述两个中英文资源文件组成的资源包,可以这样配置:

```
<message- resources parameter= "ApplicationResources"/> .
```

如果有多个资源包,那么它们用 key 属性进行区分。

```
<message- resources parameter= "ApplicationResources"/> .
<message- resources key= "Image" parameter= "ApplicationImageResources"/> .
<message- resources key= "XXX" parameter= "ApplicationXXXResources"/> .
```

其中不带 key 属性的就是默认资源包。

资源包应存放在 WEB-INT\classes 目录下，当系统初始化时，会装载这些资源文件。如果修改了资源文件，则需要重新启动应用服务器。

2. 资源文件的编码和转码

由于资源包是国际化的保证，因此它的编码应当使用标准的编码方式，一般为 UNICODE。所以，在资源包中的文件都需要经过编码转制。在 JDK 自带的工具中有一个 native2ascii 程序就是专门用来进行资源文件转码的。该工具位于 JDK 安装目录下的 bin 目录中。

需要特别注意的是，中文版属性文件 ApplicationResources_zh. properties 需要进一步处理，需要把它里面的中文字符转化为 Unicode 编码，否则国际化处理的页面将会出现乱码。利用 native2ascii 程序对中文资源文件进行编码转化的命令格式如下：

```
文件所在目录>native2ascii -encoding GB2312 ApplicationResources_zh.properties
ApplicationResources_zh_CN.properties
```

例如，假如资源文件 ApplicationResources_zh. properties 放在 D:\login\WEB-INF\classess 目录中，使用 native2ascii 对它进行转化的命令如下：

```
D:\login\WEB-INF\classess>native2ascii -encoding GB2312 ApplicationResources
_zh.properties ApplicationResources_zh_CN.properties
```

经过 native2ascii 进行转化后，生成一个新的资源文件 ApplicationResoures_zh_CN. properties，它的内容如代码清单 4-3 所示。

代码清单 4-3 中文版资源文件 ApplicationResources_zh_CN. properties

```
#\u4e00\u4e2a\u7b80\u5355\u7684\u5c5e\u6027\u6587\u4ef6

title.login=\u767B\u9646\u9875\u9762
title.register=\u6B22\u8FCE\u6CE8\u518C
title.main=\u6B22\u8FCE\u5149\u4E34\u4E3B\u9875\u9762
label.username=\u59D3\u540D
label.password=\u5BC6\u7801
label.confirm=\u786E\u8BA4\u5BC6\u7801
label.email=\u7535\u5B50\u90AE\u4EF6
item.submit=\u767B\u9646
item.reset=\u91CD\u7F6E
message.success=\u5DF2\u6210\u529F\u767B\u9646\u3002\u6B22\u8FCE\u5149\u4E34\
u3002
message.failure=\u4F60\u767B\u9646\u5931\u8D25\u3002\u73B0\u8FDB\u5165\u6CE8\
u518C\u9875\uFF0C\u8BF7\u6CE8\u518C\u4F60\u7684\u4FE1\u606F\u3002
```

3. 资源文件的访问

资源包建立好后，就可以对其进行访问了。访问的方式主要有两种：

- 在 JSP 页面中通过<bean:message>标记来访问。
- 在程序中访问。

我们在这里只讨论第一种访问资源文件的方式。

在 JSP 页面中访问资源包,需要使用<bean:message>标记。使用标记之前,必须首先部署包含此标记的 Struts 标记库文件 struts-bean.tld。具体操作步骤如下:

(1) 将 Struts 发布包中提供的标记库文件 struts-bean.tld 复制到待国际化的应用程序所在项目工程的 WEB-INF 目录下。

(2) 在使用该标记库的应用程序部署文件 web.xml 的<web-app>元素中添加相应的<taglib>元素。具体语法格式如下:

```
<taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
```

(3) 在使用<bean:message>标记的 JSP 源文件头引入此标记库,具体格式如下:

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean">
```

在使用 MyEclipse 进行开发的过程中,如果在向导中选择使用 Struts 标签,这些标签库的配置和导入都会自动完成。

更多的 Struts 标签库的使用请参考 4.2 节的内容。

4.1.3　国际化处理过程

要进行国际化处理,需要完成的开发任务依次分为如下几步:

(1) 定义资源文件。

(2) 对资源文件进行编码转化。

(3) 利用 Struts 标签对页面进行处理,访问资源文件来实现页面内容的显示。

(4) 查看国际化处理结果。

为了查看国际化的效果,需要修改 IE 浏览器的默认语言种类,然后才能看到相应的语言效果。

> **提示**:和国际化处理的显示语言有关的因素有:操作系统语言、浏览器支持的语言、资源包等因素。关于各个因素对显示界面的影响优先级如下:
>
> (1) 如果存在与本地系统编码对应的资源文件,那么如果 IE 浏览设置了本地系统支持的编码或 IE 支持的浏览器编码的对应资源文件不存在,那么系统根据本地系统编码,就会去找本地系统对应的资源文件读取。
>
> (2) 如果不存在与本地系统编码对应的资源文件:当 IE 浏览器设置了对应的编码时,回去找对应的编码资源文件,如果找不到,那么会读取默认的资源文件进行显示处理。
>
> (3) 默认资源文件的使用级别最低:依次是 IE 语言设置、本地系统设置、默认资源包。

4.2 Struts视图组件

4.2.1 Struts-html

Struts HTML标签和标准的HTML标签功能相同，我们提倡在Struts应用中使用Struts标签，是因为这些标签可以和Struts框架的其他组件紧密地联系起来，Struts能够把这个表单中的数据自动映射到对应的ActionForm中。下面对具体的标签进行深度研究。

从用户处收集数据是动态Web应用非常重要的一个方面，因此构建输入表单也就自然而然地成为Struts框架的一个重要内容。Struts HTML标签库含有创建Struts输入表单的标签，和其他标签库(bean、logic、nested和tiles)中的标签一起协作就可以产生基于HTML的用户界面。

Struts HTML标签和HTML标签之间的对应关系如表4-2。

表4-2 Struts HTML标签对照表

Struts HTML标签	HTML元素
<html:base>	<base>
<html:button>	<input type="button">
<html:cancel>	<input type="submit">
<html:checkbox>,<html:multibox>	<input type="checkbox">
<html:errors>	
<html:file>	<input type="file">
<html:form>	<form>
<html:frame>	<frame>
<html:hidden>	<input type="hidden">
<html:html>	<html>
<html:image>	<input type="image">
<html:img>	<img>
<html:javascript>	
<html:link>	<a>
<html:messages>	
<html:option><html:options><html:optionsCollection>	<option>
<html:password>	<input type="password">
<html:radio>	<input type="radio">
<html:reset>	<input type="reset">
<html:rewritte>	
<html:select>	<select>
<html:submit>	<input type="submit">
<html:text>	<input type="text">
<html:textarea>	
<html:xhtml>	

以上的 HTML 标签大致分为以下几类。

1. 用于生成基本的 HTML 元素的标签

1）<html:html>标签

用于生成 HTML 的<html>元素。它包含两个属性：

- locale：表示是否本地化，可选值为 true 或 false。
- xhtml：表示是否输出 XHTML，可选值为 true 或 false。

两者都不是必需的。例如：

```
<html:html locale="true">
```

此行代码解析后为：

```
<html lang="en">
```

生成的结果取决于 Struts 应用程序所位于的服务器的 Locale。如果你将应用程序部署到一个不同的 Locale 的服务器，则不需要改变代码，Locale 会自动调整。

2）<html:base>标签

用于生成 HTML<base>元素，表示所包含页面的绝对位置。这个标签只有内嵌在 head 标签中才有效。例如：

```
<html:base/>
```

此行代码解析后：

```
<base herf="http://www.mymain.com/myStrutsApp/testing.jsp">
```

3）<html:link>标签

用于生成 HTML 中的<a>元素，包含属性 page：表示链接地址。

```
<html:link page="/index.html">Click demo</html:link>
```

此行代码解析后：

```
<a href="/index.html">Click demo</a>
```

4）<html:rewrite>标签

用于生成用户请求的 URL，包含属性 forward：表示请求的 URL 地址。

例如：

```
<html:rewrite forward=""/>
```

5）<html:img>标签

用于生成 HTML 的<img>元素，包含以下属性。

- page：图像文件相对于模块的路径。前面必须带有一个斜线。
- height：图像的高度。
- width：图像的宽度。
- alt：如果找不到图像而显示的文本。

最重要的属性是 page。

height、width、alt 这些属性与 HTML 的<img>元素相同。例如：

```
<html:img page="/logo.gif" height="50" width="200" alt="Web Logo"/>
```

此行代码解析后：

```
<img src="/logo.gif" height="50" width="200" alt="Web Logo">
```

2. 用于生成 HTML 表单的标签

1）<html:form>标签

用于生成 HTML 的<form>元素。包含的属性如下。

- method：属性的默认值是 POST。
- action：是这个标签中唯一必需的属性。如果不具备该属性，则 JSP 页面会抛出一个异常。之后你必须给这个 action 属性指定一个有效值。一个有效值是指应用程序的 Struts 配置文件中元素里的任何一个子元素的访问路径。而且相应的元素中必须有一个 name 属性，它的值是 Form Bean 的名称。

例如：

```
<html:form action="userAction.do">
```

那么，Struts 配置文件的元素中必须有一个显示为粗体的元素，如下：

```
<action
        attribute="loginForm"
        input="/login.jsp"
        name="userForm"
        path="/userAction"
        scope="request"
        type="com.wangyingling.struts.actions.UserAction"
        parameter="method">
        <forward name="welcome" path="/main.jsp" />
        <forward name="fail" path="/login.jsp" />
        <forward name="login" path="/login.jsp" />
        <forward name="exit" path="/login.jsp" />
</action>
```

任何包含在<form>中用来接收用户输入的标签(<text>、<password>、<hidden>、<textarea>、<radio>、<checkbox>、<select>)必须在相关的 form bean 中有一个指定的属性值。比如，如果你有一个属性值被指定"username"的<text>标签，那么相关的 Form Beam 中必须有一个名为"username"的属性。输入<text>标签中的值会被用于生成 From Bean 的 userName 属性。

2）数据输入标签

主要包括如下。

- <html:text>标签：用于生成 HTML 的<input type="text">元素。

- <html:password>标签：用于生成 HTML 的<input type="password">元素。
- <html:hidden>标签：用于生成 HTML 的<input type="hidden">元素。
- <html:textarea>标签：用于生成 HTML 的<input type=/textarea>元素。
- <html:radio>标签：用于生成 HTML 的<input type="radio">元素。
- <html:checkbox>标签：用于生成 HTML 的<input type="checkbox">元素。
- <html:select>标签：用于生成 HTML 的<input type="select">元素。
- <html:option>标签：用于生成 HTML 的<input type="option">元素。

<html:password>标签中的一个很重要的属性是"redisplay"，它用于重新显示以前输入到这个区域中的值。该属性的默认值为 true。然而，为了使 password 不能被重新显示，你或许希望将该属性的值设为 false。

3）提交按钮标签

主要包括如下。

- <html:submit>标签：用于生成 HTML 的<input type="submit">元素。
- <html: submit>标签：用于在表单上生成取消按钮。
- <html:reset>标签：用于生成 HTML 的<input type="reset">元素。
- <html:button>标签：用于生成 HTML 的<input type="button">元素。
- <html:image>标签：用于生成 HTML 的<input type="image">元素。

3. 用于显示错误或正常消息的标签

通过一个简单的<html:errors/>标签，你就可以在一个 JSP 页面上显示完全自定义的错误信息。该标签将由 name 属性指定的 ActionMessages、ActionErrors、String 和 String[]直接输出到页面中。

如果在应用程序资源中存在相应的信息，那么就可以用下面这些可选的 message keys。

- errors. header：相应的信息在错误信息的单独列表前显示。
- errors. footer：相应的信息在错误信息的单独列表后显示。
- errors. prefix：相应的信息在错误信息的单独列表前显示。
- errors. suffix：相应的信息在错误信息的单独列表后显示。

4.2.2 Struts-bean

此标签库和 JavaBean 有很强的关联性，设计的本意是要在 JSP 和 JavaBean 之间提供一个接口。Struts 提供了一套小巧有用的标签库来操纵 JavaBean 和相关对象。可以将 Bean 标签库分为以下三大类。

1. 用于访问 HTTP 请求信息或 JSP 隐含对象

这一类共包含 4 个标签。

1）<bean:cookie>：用于访问 Cookie 信息

Cookie 最早是由 Netscape 公司提出来的，用来存储客户的少量状态信息。

<bean:cookie>标签取回请求中名称为 name 的 cookie 的值。如果没有指定 multiple 属性则依据刚取回的值创建一个 Cookie 类型的 bean。如果指定了 multiple 属

性则依据刚取回的值创建一个 Cookie[]类型的数组。然后用 id 属性值将 Cookie 或 Cookie[]绑定到 page 作用域中,并创建对应的脚本变量。

2）<bena:header>：用于访问 HTTP 请求中的 Header 信息

<bena:header>标签取回请求中名称为 name 的 header 的值。如果没有指定 multiple 属性则依据刚取回的值创建一个 String 类型的 bean。如果指定了 multiple 属性则依据刚取回的值创建一个 String[]类型的数组。然后用 id 属性值将 String 或 String[]绑定到 page 作用域中,并创建对应的 scripting 变量。

3）<bean:parameter>：用于访问请求参数

<bean:parameter>标签取回请求中的参数值。如果没有指定 multiple 属性则依据刚取回的值创建一个 String 类型的 bean。如果指定了 multiple 属性则依据刚取回的值创建一个 String[]类型的数组。然后用 id 属性值将 String 或 String[]绑定到 page 作用域中,并创建对应的 scripting 变量。

下面两个代码片段使用相同的 url 传递参数,url 的形式为 http://127.0.0.1:8080/struts-demo/paramter/jsp? param=1&param=2&param=3。前面的代码片段中没有指定 multiple 属性,因此 p 是 String 类型而且仅仅读取了参数的第一个值。后面的代码片段中指定了 mulitiple 属性的值,因此 ps 是 String[]类型的包含所有参数的值。

```
<bean:parameter id="p" name="param" />
    <bean:write name="p" />
    <bean:parameter id="ps" multiple="true" name="param" />
    <logic:iterate id="p" name="ps">
        <bean:write name="p" />
</logic:iterate>
```

4）<bean:page>：用于访问 JSP 隐含对象

<bean:page>标签将页面上下文中的 application、config、request、response 或 session 取出,然后用 id 属性值将它们绑定到 page 作用域中,并创建对应的 scripting 变量。

下面的代码片段演示了如何使用<bean:page>标签取出 response,然后使用<bean:write>标签将 response 的 charaterEncoding 和 contentType 属性输出。

```
<!--bean:page 标签-->
<bean:page id="res" property="response"/>
<!--输出绑定 bean 对象-->
<bean:write name="res" propetry="characterEncoding"/>
<bean:write name="res" propetry="contentType"/>
```

你可以用和上面类似的代码访问 application、config、request 或 session 中的任何一个对象。

2. 用于访问 Web 应用资源

这里我们仅介绍<bean:message>标签的使用。

<bean:message>标签用来从指定的 locale 中取回国际化的消息并输出。在这个过程中我们还可以传递 5 个以内的参数。message key 可以通过 key 直接指定，也可以通过 name 和 property 间接指定。

<bean:message>标签有两种指定 message key 的方式，一是通过 key 属性直接指定；二是通过 name 和 property 属性间接指定，其中 message key 是在 message resources 文件中定义的。

为了介绍该标签这里使用了三个 message resources 文件，文件的名字分别为 ApplicationResources. properties、ApplicationResources_en. properties、ApplicationResources_zh_CN. properties，在 struts-config. xml 文件中的设置如下。

```
<message- resources parameter= "ApplicationResources"/>
```

3. 用于定义或输出 JavaBean 的 Bean 标签

这一类共包含三个标签。

1）<bean:define>：用于定义一个变量

<bean:define>标签在 toScope（如果没有指定值就使用 page 作用域）指定的作用域中创建一个新属性，同时创建一个 scripting 变量。我们可以通过 id 值使用它们。新创建的属性可以由其他标签使用，新创建的 scripting 变量可以由 JSP 脚本使用。

可以使用三种方式为新创建的属性和 scripting 变量赋值。

- 通过该标签的 name、property 和 scope 取回值，并且保持类型的一致性，除非取回的值为 Java 的原始类型，这时会使用适合的包装器类对这些值进行包装。
- 通过该标签的 value 指定值，这时新创建的属性和 scripting 变量的类型为 java. lang. String。
- 在该标签体中嵌入值，这时新创建的属性和 scripting 变量的类型为 java. lang. String。

下面的代码片段演示了如何使用<bean:define>标签创建新属性 values 和新 scripting 变量 values，它将 listForm 值 persons 的值取出来赋给 values。

```
<bean:define id= "values" name= "listform" property= "persons" type=
"java.util.List"/>
```

2）<bean:write>：用于显示 JavaBean 或其属性的内容

<bean:write>标签将指定的 bean 的属性值写到当前的 JspWriter 中，并且可以对输出进行格式化。

下面的代码片段演示了如何使用<bean:write>标签格式化输出当前日期，其中 now 是在 DataForm 中定义的一个 java. util. Date 类型的域（值为 new Date()），format. date. standard 是在资源文件值的一个键（format. date. standard＝yyyy-MM-dd）。

```
<bean:define id= "date" name= "dataForm" property= "now">< /bean:define>
<bean:write name= "date"/>
<bean:write name= "date" format= ""MM/dd/yyyy/>
```

上面代码运行的结果为：

```
Sun Jun 04 17:04:05 CST 200606/04/20062006-06-04
```

3）<bean:size>：用于获得 Map 或 Collection 集合的长度

<bean:size>标签创建一个 java.lang.Interger 类型的 bean，这个 bean 的值为该标签指定的 Collection 或 Map 值所含元素的个数。这可以和<logic:iterate>标签配合使用，因为<logic:iterate>标签不能得到所迭代的集合的元素个数，这有时候很不方便。

4.2.3 Struts-logic

Struts 的 Logic 标签可以根据特定的逻辑条件来判断网页的内容，或者循环遍历集合元素，它和 HTML、Bean 标签是 Struts 应用中最常用的三个标签。它的主要功能是：

- 比较运算。
- 进行字符串的匹配。
- 判断指定的内容是否存在。
- 循环遍历集合。
- 进行请求转发和重定向。

下面就分别来研究这 5 种功能的标签。

1. 进行比较运算的 Logic 标签

具有该功能的标签共包含 6 个。

- <logic:equal>(=)：比较变量是否等于指定的常量。
- <logic:notEqual>(=)：比较变量是否不等于指定的常量。
- <logic:greaterEqual>(>=)：比较变量是否大于或等于指定的常量。
- <logic: greaterThan>(>)：比较变量是否大于指定的常量。
- <logic: Equal>(<=)：比较变量是否小于或等于指定的常量。
- <logic: lessThan>(<)：比较变量是否小于指定的常量。

以上这些标签都很类似，它们拥有如下共同的属性。

- cookie 属性指定 cookie 属性的值，然后用 value 设置的常量进行比较。
- header 属性设置 header 请求的值，也是通过 value 属性设置的值进行比较。
- parameter 属性设置一个请求参数，然后也是通过 value 属性设置的值进行比较。
- name 属性设置一个变量，然后用 value 比较。如果同时设置了 name 和 property 属性，此时 name 属性指定已存在的 JavaBean，property 属性指定 Bean 的属性。

2. 进行字符串匹配的 Logic 标签

下面研究一下进行字符串匹配的标签和判断特定内容的标签。

- <logic:match>：判断变量值是否包含指定的常量字符串。
- <logic:notmatch>：判断变量值是否不包含指定的常量字符串。

用 name 属性定义一个字符串变量，然后用 value 属性的值去判断变量是否包含这个字符串或者是不包含这个字符串，判断成功返回 true。

3. 判断指定内容是否存在的 Logic 标签

下面来看判断指定内容是否存在 Logic 标签。

(1) <logic:empty>和<logic:notEmpty>标签：用于判断指定的变量是否为空的字符串，通过 name 属性来判断一个字符串是否为 null。

<logic:empty>标签是用来判断是否为空的。如果为空，该标签体中嵌入的内容就会被处理。该标签用于以下情况。

- 当 Java 对象为 null 时。
- 当 String 对象为“”时。
- 当 java.util.Collection 对象中的 isEmpty()返回 true 时。
- 当 java.util.Map 对象中的 isEmpty()返回 true 时。

<logic:notEmpty>标签的应用正好和<logic:empty>标签相反。

(2) <logic:present>和<logic:notPresent>标签：用于判断指定的对象是否存在。

这个标签的属性很多，如下：

- cookie 属性判断 Cookie 是否存在。
- Header 属性判断 HTTP 请求头是否存在。
- role 属性判断当前的权限的用户是否是指定的安全角色。
- user 属性判断当前通过权限验证的用户是否拥有指定的用户名。
- parameter 属性判断请求的参数是否存在。
- name 属性用来判断指定的 Bean 值是否存在。同时设置 name 和 property 属性就是判断 Bean 中的具体属性是否存在了。

如果指定的值出现该标签，就会创建其具体标签内的内容。该标签用于以下情况：

- 检查具有指定名称的 cookie 是否出现。
- 检查具有指定名称的 header 是否出现。
- 检查具有指定名称的 JSP Bean 是否出现，或者检查具有指定名称的 JSP Bean 中的 property 属性是否出现。
- 检查 request 中指定名称的参数是否出现。
- 检查当前的认证用户是否与指定的安全角色相关联。
- 检查当前认证的主体是否具有指定的名字。

<logic:notPresent>标签的应用正好和<logic:present>标签相反。

(3) <logic:messagePresent>和<logic:messageNotPresent>标签：用来判断是否存在 request 范围内特定的 ActionMessages 或者是子类 ActionErrors 对象。

它含有两个属性：

- Name 属性用来检索 ActionMessages 对象的 key。
- Property 属性指定从 ActionMessages 集合对象中检索某条消息的 key，也就是具体的 ActionMessage 对象。

<logic:messagePresent>标签用于下列情况：

- 在 request 作用域中存在一个 ActionMessages 对象，标签的 property 属性和 ActionMessages 中的 property 属性对应。
- 在 request 作用域中存在一个 ActionErrors 对象，标签的 property 属性和

ActionErrors 中的 property 属性对应。

- 存在一个 String 对象，将其转换成 ActionMessage 然后添加到 ActionMessages 中。
- 存在一个 String Array 对象，将数组中的每一个 String 对象转换成一个 ActionMessage，然后将其添加到 ActionMessages 中。

标签的 message 属性值为 true 时，将以 Globals. MESSAGE_KEY 为 key 在 request 作用域中查找 Message，其他情况下，将 name 值作为 key 查找，如果 name 没有出现，默认为 Globals. ERROR_KEY。

<logic: messgesPresent>标签的应用正好和<logic: messgesNotPresent>标签相反。

4. 进行循环遍历的 logic 标签

进行循环遍历的 logic 标签是 logic 标签库中最复杂的标签，也是用途最广泛的标签。它能够在一个循环中遍历数组、Collection、Enumeration、Iterator 或者 Map 中的所有元素。

<logic:iterate>包含的属性包括：

- name 属性指定需要进行遍历的集合对象，它每次从集合中检索出一个元素，然后放在 page 范围内。
- id 属性指定这个字符串来命名这个元素，最好是在一个里面嵌套一个<bean:write>标签，把刚刚遍历的第一个 id 指定的字符串输出，然后再循环输出。
- length 属性指定需要遍历的元素的数目，如果没有设置 length 属性，就遍历集合中的所有元素。
- offset 属性指定开始遍历的起始位置，默认值是 0
- indexld 属性定义一个代表当前遍历元素的系列号，这个变量存在 page 范围内，可以被<logic:write>标签访问输出的是 int 的数字，如 1、2、3、4 等。

上面提到的集合可以是：

- 对象类型或原子类型的数组(Array)。
- java. util. Collection 的实现，包括 ArrayList、Vector。
- java. util. Enumeration 的实现。
- java. util. Iterator 的实现。
- java. util. Map 的实现，包括 HashMap、Hashtable 和 TreeMap。

如果你迭代的集合中含有 null 的值，这时需要采取一定的措施，因为这时<logic:iterate>不会在 page 作用域中创建对象。一般使用<logic:present>标签或<logic:notPresent>标签来判断一下。

5. 进行请求转发或重定向的 logic 标签

进行请求转发或重定向的 logic 标签如下：

1) <logic:forward>进行请求转发

<logic:forward>标签用于请求转发，它的 name 属性指定转发的目标，与 struts 配置文件中的<global-forward>元素和子元素<forward>匹配。简单地说就是<logic:forward>标签的 name 属性定义的值，要去找到<global-forwards>子元素<forward>匹配的 name 属性，然后通过 path 指定的路径进行转发。

2）<logic:redirect>进行请求重定向

重定向用<logic:redirect>标签指定它的 forward、href 和 page 属性指定重定向的目标，这几个属性和<html:link>标签的属性用法十分相似。

然后我们看看这两个动作的区别。

- forward 是在 Servlet 内部执行的，浏览器完全不会感知到这个动作，原始的 URL 也不会改变，浏览器重新装载的话也是对原始的请求进行简单的重复。
- redirect 则分成两个步骤：第一步是 Web 应用程序告诉浏览器第二个 URL，然后浏览器向第二个 URL 发送请求。

Redirect 比 forward 慢，因为浏览器要做第二次请求，还有就是要注意，在第一次请求作用域内的 bean 对于第二次请求是不可见的。

4.3　为原型系统添加国际化处理

我们要对前面实现的原型功能进行国际化的支持，要求支持中文和英文。需要修改的组件有：

- 资源文件。
- 配置文件：struts-config. xml。
- 视图组件：login. jsp、register. jsp、main. jsp，分别如代码清单 4-4～代码清单 4-6 所示。

具体步骤如下：

（1）创建所需资源包。在 Web 项目的 com. wangyingling. struts 目录下创建三个属性文件 ApplicationResources. properties、ApplicationResources _ en. properties 和 ApplicationResources_zh. properties。前两个资源文件分别为默认资源文件和英文资源文件，其内容一样，如代码清单 4-1，最后一个资源文件为中文资源文件，内容如代码清单 4-2。

（2）资源文件转码。利用 native2ascii 程序把 ApplicationResources_zh. properties 文件转码成 UNICODE 编码的文件，存放在 com. wangyingling. struts 目录下，并命名为 ApplicationResources_zh_CN. properties。转码命令为：

```
文件所在目录>native2ascii -encoding GB2312 ApplicationResources_zh.properties
ApplicationResources_zh_CN.properties
```

（3）修改 struts-config. xml。在原来的 struts-config. xml 中添加<message-resource>元素，以指明要用到的资源包：

```
<message-resources parameter="com.wangyingling.struts.ApplicationResources" />
```

（4）用视图标签修改 Struts 视图。

代码中的黑体字为新增内容，用<bean:message>读取对应的标签内容替换原来页面中的静态文本实现国际化。

代码清单 4-4　添加了国际化支持的登录页面 login. jsp

```
<%@page language="java" pageEncoding="UTF-8"%>
```

```
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>

<html>
    <head>
        <title><bean:message key="title.login"/></title>
    </head>
    <body>
        <html:form action="/userAction?method=doLogin">
            <bean:message key="label.username"/>:<html:text property="uname" />
            <html:errors property="uname" />
            <br />
            <bean:message key="label.password"/>:<html:password property=
            "upass" />
            <html:errors property="upass" />
            <br />

            <html:submit/>
            <html:cancel />
        </html:form>
            <html:link target="_self" href="register.jsp"><bean:message key=
            "label.href.register"/></html:link>
    </body>
</html>
```

代码清单 4-5 添加了国际化支持的注册页面 register.jsp

```
<%@ page language="java" pageEncoding="GBK"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<html>
    <head>
        <title><bean:message key="title.register"/></title>
    </head>
    <body>
        <html:form action="/userAction?method=doRegister">
        <bean:message key="label.username"/><html:text property="uname" />
            <html:errors property="uname" />
            <br />
            <bean:message key="label.password"/><html:password property=
            "upass2" />
            <html:errors property="upass2" />
            <br />
            <bean:message key="label.confirm"/><html:password property=
            "upass" />
            <html:errors property="upass" />
```

```
            <br />
            <bean:message key="label.email"/><html:text property="email" />
            <html:errors property="email" />
            <br />

            <html:submit />
            <html:cancel />
        </html:form>
    </body>
</html>
```

代码清单 4-6　添加了国际化支持的主页面 main.jsp

```
<%@page language="java" import="java.util.*" pageEncoding="GB18030"%>
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@taglib uri="http://struts.apache.org/tags-html" prefix="html"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title><bean:message key="title.main"/></title>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
    </head>
    <body>
        <div align="center">
            <font size="4"><strong><font color="#008000">
                        ${sessionScope.userName }</font><bean:message key=
                        "message.success"/><br><br>
                <form action="userAction.do?method=doExit" method="post">
                    <input type="hidden" value="logout" name="opType">
                    <input type="submit" name="submit" value="退出">
                </form><br></strong></font>
        </div>
    </body>
</html>
```

(5) 部署运行程序，运行结果如图 4-1～图 4-3 所示。

- 发布并运行 strutsDemo 项目。
- 打开 IE 浏览器，通过“工具”→“Internet 选项”菜单中的“语言栏”，改变默认语言为中文(zh_cn)，运行三个页面可以看到如图 4-1～图 4-3 所示的效果。
- 修改 IE 浏览器的默认语言为英文(en)，运行三个页面可以看到如图 4-4～图 4-6 所示的效果。



图 4-1 浏览器默认语言为中文的登录页效果



图 4-2 浏览器默认语言为中文的注册页效果



图 4-3 浏览器默认语言为中文的主页效果



图 4-4 浏览器默认语言为英文时登录页效果

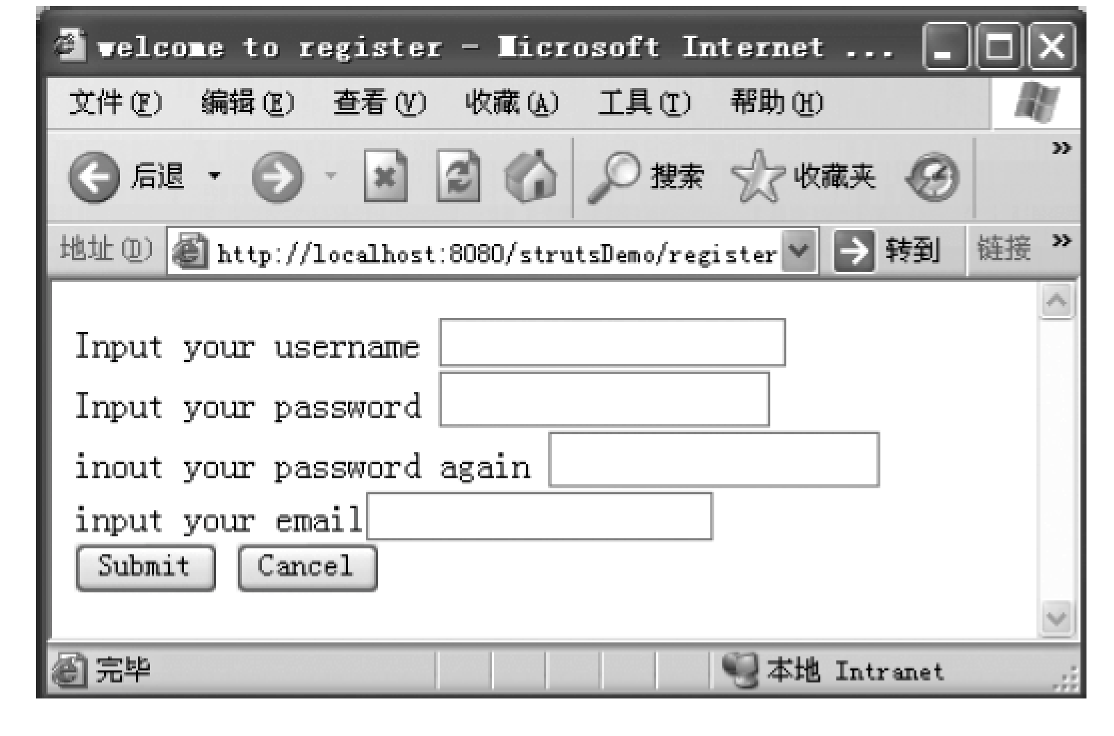


图 4-5 浏览器默认语言为英文时注册页效果



图 4-6 浏览器默认语言为英文时主页效果

4.4　实验与能力拓展

1. 请根据教材内容完成国际化处理练习。

2. 请运行国际化支持的原型系统，并通过修改浏览器的默认语言来理解国际化的运行原理。

3. 假定资源包中只有默认资源文件 ApplicationResources. properties 和中文资源文件 ApplicationRsources_zh_CN. properties，IE 的默认浏览器语言是英文，操作系统是中文版本，请推测浏览器的浏览结果是什么语言内容的？并结合运行原理思考为什么。

第 5 章　Hibernate 入门

本章导读

本章开始将介绍 MVC 架构的模型层的实现框架——Hibernate。从介绍 Hibernate 是什么开始，逐步介绍 Hibernate 开发中要重点关注的和常用的一些核心组件，然后是开发 Hibernate 应用的步骤，并通过实现原型系统中的注册功能来实现 Struts 和 Hibernate 的结合。

工作任务

改造原型系统中的注册功能的模型层实现，使用 Hibernate 实现对数据库的访问操作。

5.1　Hibernate 介绍

5.1.1　基础理论

ORM(Object Relational Mapping)是指对象关系映射。它是一种为了解决面向对象与关系数据库存在的互不匹配现象的技术。简单地说，ORM 是通过使用描述对象和数据库之间映射的元数据，将 Java 程序中的对象自动持久化到关系数据库中。本质上是将数据从一种形式转换为另外一种形式。

图 5-1 通过用户实体、数据表、Java 类三者之间的映射关系，反映了 ORM 在程序开发中的作用。

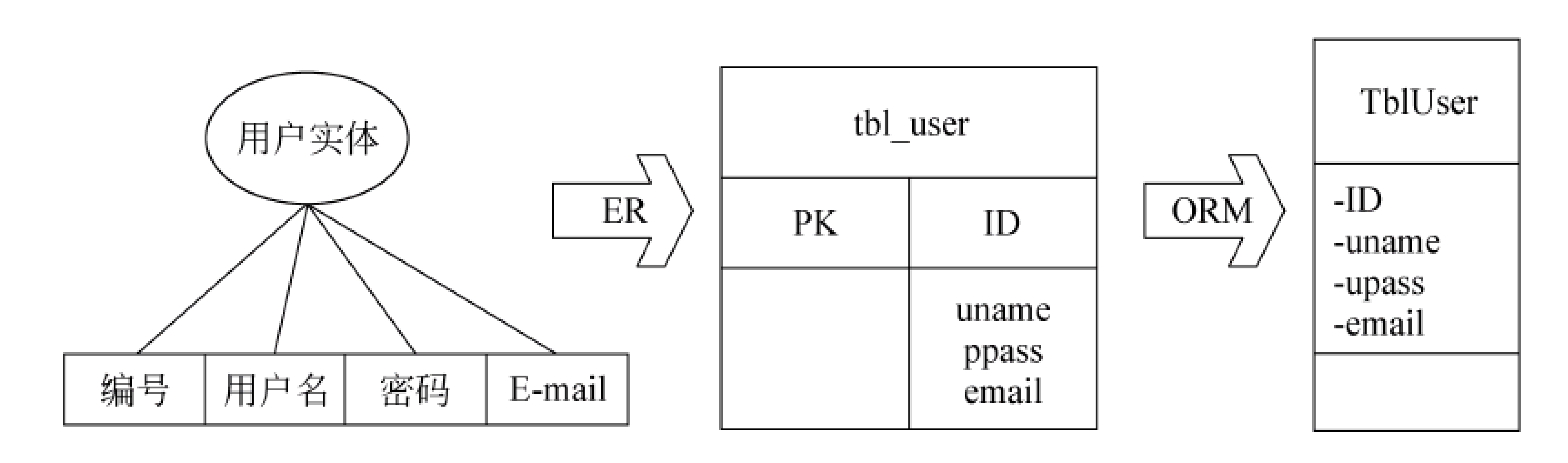


图 5-1　ORM 示意图

该图中依次包含了数据实体的三种不同表示形式，即数据实体、数据表和映射对象（实体类）。数据表是通过“实体-关系”即 ER 映射，从数据实体映射为数据表的；映射对象是通过“对象-关系”即 ORM 映射，从数据表映射为实体对象的。

ORM 是通过使用描述对象和数据库之间映射的元数据，将 Java 程序中的对象自动持久化到关系数据库中。因此我们需要理解两个概念：

- **持久化（Persistence）**：就是对数据和程序状态的保持，也就是说把数据从瞬时状态转化到永久状态。瞬时状态指的是数据在断电后就会丢失的状态，而持久状态指数据在断电前后能保持不变的状态。大多数情况下特别是企业级应用，数据持久化往往也就意味着将内存中的数据保存到磁盘上加以固化，持久化的实现过程大多通过各种关系数据库来完成。
- **持久层（Persistence Layer）**：把数据库实现作为一个独立逻辑拿出来，即数据库程序是在内存中的，为了使程序运行结束后状态得以保存，就要保存到数据库。持久层是在系统逻辑层面上，专注于实现数据持久化的一个相对独立的领域。

持久层的目的是通过持久层的框架将数据库存储从服务层中分离出来。Hibernate 是目前最流行的持久层框架，它是对 JDBC 的轻量级的对象封装，可以用在任何 JDBC 可以使用的场合，例如 Java 应用程序的数据库访问代码、DAO 接口的实现类等。

5.1.2 Hibernate 简介

Hibernate 是对象/关系映射（Object Relational Mapping，ORM）的解决方案，简单地说，就是 Java 对象与对象关系映射至关系型数据库中的数据表与数据表之间的关系，Hibernate 提供了这个过程中自动对应转换的方案。

Hibernate 是 Java 应用和关系数据库之间的桥梁，它负责对 Java 对象和关系数据库之间的映射。Hibernate 内部封装了通过 JDBC 访问数据库的操作，向上层应用提供了面向对象的数据库访问 API。在基于 MVC 设计模式的 Java Web 应用中，Hibernate 可以作为应用的数据访问层或持久层。它具有以下特点：

（1）Hibernate 的目标是成为 Java 中管理数据库持久性问题的一种完整解决方案。它协调应用与关系数据库的交互，让开发者解放出来专注于手中的业务问题。

（2）Hibernate 是一种非强迫性的解决方案。开发者在写业务逻辑与持久化类时，不会被要求遵循许多 Hibernate 特定的规则和设计模式。这样，Hibernate 就可以与大多数新的和现有的应用平顺地集成，而不需要对应用的其余部分做破坏性的改动。

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非轻量级的对象封装，使得 Java 程序员可以随心所欲地使用面向对象编程思想来操作数据库。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序使用，也可以在 Servlet/JSP 的 Web 应用中使用，最具革命意义的是，Hibernate 可以在 J2EE 中取代 CMP，完成数据持久化的重任。

5.2 Hibernate 应用框架

5.2.1 Hibernate 体系结构

在基于 MVC 设计模式的 Java Web 应用中，Hibernate 可以作为模型层/数据访问层。它通过配置文件(hibernate. properties 或 hibernate. cfg. xml)和映射文件(hbm. xml)把 Java 对象或持久化对象(Persistent Object，PO)映射到数据库中的数据表，然后通过操作 PO，对数据库表中的数据进行增、删、改、查操作。Hibernate 的体系结构如图 5-2 所示。

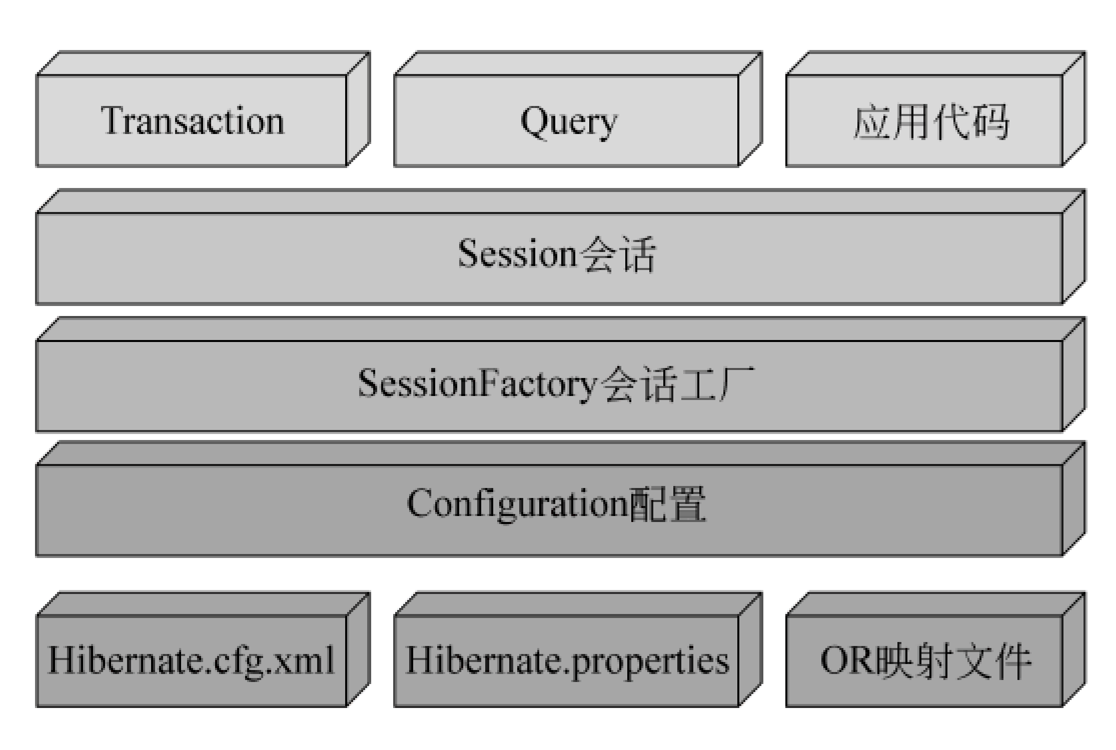


图 5-2 Hibernate 的体系结构

5.2.2 Hibernate 核心组件

除配置文件(hibernate. properties 或 hibernate. cfg. xml)、映射文件(hbm. xml)和持久化类外，Hibernate 的核心组件包括以下几部分：

(1) Configuration 类：用来读取 Hibernate 配置文件，并生成 SessionFactory 对象。

(2) SessionFactory 类：产生 Session 实例的工厂。

(3) Session 接口：用来操作 PO。它有 get()、load()、save()、update()和 delete()等方法来对 PO 进行加载、保存、更新及删除等操作。它是 Hibernate 的核心接口。

(4) Query 接口：用来对 PO 进行查询操作。它可以从 Session 的 createQuery()方法生成。

(5) Transaction 接口：用来管理 Hibernate 事务，它的主要方法有 commit()和 rollback()，可以从 Session 的 beginTransaction()方法生成。

Hibernate 配置文件主要用来配置数据库连接参数，例如数据库的驱动程序、URL、用户名和密码等。Hibernate 支持两种格式：hibernate. properties 和 hibernate. cfg. xml。两者的配置内容基本相同，但后者的使用稍微方便一些，我们所采用的开发工具中默认生成的配置文件就是 hibernate. cfg. xml，例如 hibernate. cfg. xml 可以在其<mapping>子元素中定义用到的 xxx. hbm. xml 映射文件列表，而使用 hibernate. properties 则需要在程序中以硬编码的方式指明。在一般情况下，hibernate. cfg. xml 是 Hibernate 的默认配

置文件。

映射文件用来把 PO 与数据库中的表、PO 之间的关系与数据库表之间的关系、一级 PO 的属性与表字段一一映射起来，它是 Hibernate 的核心文件。

持久化对象(Persistent Objects，PO)可以是普通的 JavaBean，唯一特殊的就是它们与(仅一个) Session 关联。JavaBeans 在 Hibernate 中存在三种状态：临时状态(transient)、持久化状态(persistent)和脱管状态(detached)。当一个 JavaBean 对象在内存中孤立存在、不与数据库中的数据有任何关联关系时，那么这个 JavaBean 对象就称为临时对象；当它与一个 Session 相关联时，就变成持久化对象；在这个 Session 被关闭的同时，这个对象也会脱离持久化状态，变成脱管对象，可以被应用程序的任何层自由使用，例如可用作与表示层打交道的数据传输对象。

5.3 Hibernate 核心

5.3.1 Hibernate 配置文件

Hibernate 支持 Java 属性和 XML 两个格式的配置文件。

Hibernate 支持的属性文件必须符合 Java 属性文件的规范，配置信息均以“键＝值”的形式存在，每条配置信息应单独占一行。使用“＃”可以为属性文件添加注释。通常属性配置文件应被命名为 hibernate. properties，并保存在系统类路径根目录下。

使用属性配置文件往往需要必要的编码工作来进一步导入对象映射元数据。而使用 Hibernate 支持的 XML 风格的配置文件时，由于对象映射信息的数据源可以直接在 hibernate. cfg. xml 中进行配置，Hibernate 将全权负责导入配置和对象映射信息的工作，因此开发人员编码的工作量相对比较少。

XML 配置文件 hibernate. cfg. xml 全面直观，所有在属性文件中配置的信息都可以完全相等地在 hibernate. cfg. xml 中进行配置。本书只讨论 XML 格式的配置文件。代码清单 5-1 是一个典型的 XML 配置文件，包含必需的配置标签和可选标签。

代码清单 5-1 hibernate. cfg. xml 典型配置

```
<?xml version='1.0' encoding='gb2312'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!--显示执行的 SQL 语句-->
        <property name="show_sql">true</property>
        <!--连接字符串-->
        <property name="connection.url">jdbc:mysql://localhost:3306/mvcDB
        </property>
        <!--连接数据库的用户名-->
```

```
        <property name="connection.username">root</property>
        <!--数据库用户密码-->
        <property name="connection.password">root</property>
        <!--数据库驱动-->
        <property name="connection.driver_class">com.mysql.jdbc.Driver
        </property>
        <!--选择使用的方言-->
        <property name="dialect">org.hibernate.dialect.MySQLDialect
        </property>
      <!—指定要加载的映射文件-->
        <mapping resource="events/Event.hbm.xml"/>
        <mapping resource="events/Person.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

其中各个标签元素的含义已经在配置文件中作了注释说明。其中<mapping>标签用来指定要加载的映射文件，resource属性指出要加载的映射文件的位置。

5.3.2 Hibernate映射文件

一个应用程序的实体类好比一篇文章中的名词，通常只包含业务逻辑需要的具体信息，如用户的姓名、密码和电子邮件等。如果希望将实体对象保存到数据库表格中，ORM还需要进一步映射信息，例如User对象的uname属性将保存到数据库tbl_user表格的uname列中。这类描述对象映射的数据称为映射元数据。

Hibernate也支持两种形式的映射元数据，其中一种是XML文件。映射文件主要描述实体类和数据表的对应关系，以及实体类的属性和表的列之间的对应关系等。

本书只讨论XML格式的映射文件。假设有一个实体类TblUser.java，其内容如代码清单5-2，则与之对应的映射文件名为TblUser.hbm.xml，内容如代码清单5-3，定义了User对象的各个属性的映射信息。

代码清单5-2 TblUser.java

```
package com.wangyingling.hibernate.beans;
public class TblUser implements java.io.Serializable {
    //属性
    private Integer id;
    private String uname;
    private String upass;
    private String email;
    //属性的 get()和 set()方法
    public Integer getId(){
        return this.id;
    }
    public void setId(Integer id){
        this.id=id;
```

```
    }
    public String getUname(){
        return this.uname;
    }
    public void setUname(String uname){
        this.uname= uname;
    }
    public String getUpass(){
        return this.upass;
    }
    public void setUpass(String upass){
        this.upass= upass;
    }
    public String getEmail(){
        return this.email;
    }
    public void setEmail(String email){
        this.email= email;
    }
    //构造器
    /** default constructor * /
    public TblUser(){
    }
    /** minimal constructor * /
    public TblUser(String uname, String upass){
        super(uname, upass);
    }
    /** full constructor * /
    public TblUser(String uname, String upass, String email){
        super(uname, upass, email);
    }
}
```

代码清单 5-3 TblUser. hbm. xml

```
<?xml version= "1.0" encoding= "utf- 8"?>
<!DOCTYPE hibernate- mapping PUBLIC "- //Hibernate/Hibernate Mapping DTD 3.0
//EN"
"http://hibernate.sourceforge.net/hibernate- mapping- 3.0.dtd">
<!- -
    Mapping file autogenerated by MyEclipse Persistence Tools
- ->
<hibernate- mapping>
    <class name= "com.wangyingling.hibernate.beans.TblUser" table=
    "tbl_user" catalog= "mvcdb">
        <id name= "id" type= "java.lang.Integer">
```

```
            <column name="id" />
            <generator class="native" />
        </id>
        <property name="uname" type="java.lang.String">
            <column name="uname" length="50" not-null="true" />
        </property>
        <property name="upass" type="java.lang.String">
            <column name="upass" length="20" not-null="true" />
        </property>
        <property name="email" type="java.lang.String">
            <column name="email" length="50" />
        </property>
    </class>
</hibernate-mapping>
```

其中各个标签元素的含义如下：

- <hibernate-mapping>是映射文件的根元素。每一个 hbm. xml 文件都有唯一的一个根元素。
- 使用<class>定义类。<class>是<hibernate-mapping>的子元素，用以定义一个持久化类与数据表的映射关系。
- 使用<id>定义主键。在关系数据库中，主键用来识别记录，并保证每条记录的唯一性。
- 使用<generator>设置主键生成方式。该元素的作用是指定主键生成器。它通过一个 class 属性指定生成器对应的类。
  - ◆ assigned 表示主键由外部程序负责生成，无需 Hibernate 参与，因此需要应用程序在执行 save()之前为对象分配一个标识符。这是<generator>元素没有指定时的默认生成策略。
  - ◆ native 表示由 Hibernate 根据底层数据库自行判断采用 identify、hilo 和 sequence 其中的一种作为主键生成方式。
- 使用<property>定义属性。<class>元素中，除了要包含<id>之外，最多的还是<property>。<property>元素为类定义了一个持久化的 JavaBean 风格的属性。其中 name 定义属性的名字，是必选的；column 是对应的数据库字段名，是可选的。

在保存 User 对象时，对象中的属性将会被保存到数据库 tbl_user 表格对应列中。

使用 XML 可以分离实体类和映射元数据，从而在不改变 Java 源代码的情况下可以修改对象与数据库表格的映射关系。

映射文件应该遵循下列规则：

(1) 映射文件名和对应的实体类名一致。

(2) 映射文件和对应的实体类保存在同一个位置。

(3) 映射文件必须在 hibernate. cfg. xml 文件中加载。通过<mapping>标签的 resource 标签可以实现加载。

5.3.3 Hibernate 运行原理

Hibernate 的运行过程如下：

(1) 应用程序先调用 Configuration 类，该类读取 Hibernate 配置文件及映射文件中的信息；

(2) Configuration 类利用配置和映射信息生成一个 SessionFactory 对象；

(3) SessionFactory 对象生成一个 Session 对象；

(4) Session 对象生成一个 Transaction 对象；

(5) Session 对象通过 get()、load()、save()、update、delete()和 saveOrUpdate()等方法对 PO 进行加载、保存、更新、删除等操作；在查询的情况下，可通过 Session 对象生成一个 Query 对象，然后利用 Query 对象执行查询操作；如果没有异常，Transaction 对象将提交这些操作结果到数据库中。

Hibernate 的运行过程如图 5-3 所示。

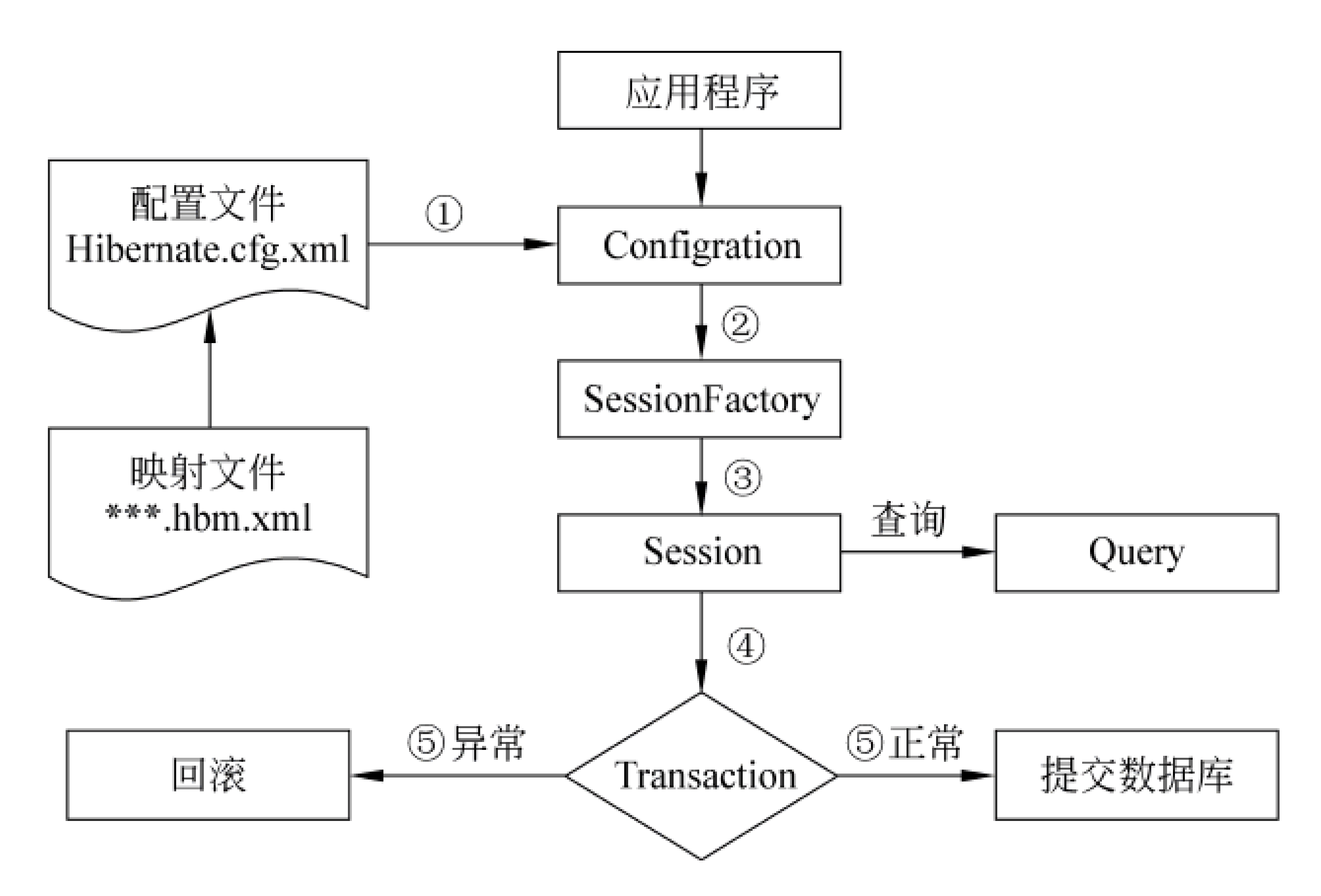


图 5-3 Hibernate 的运行原理图

5.4 应用 Hibernate 开发原型中的注册功能

5.4.1 Hibernate 应用开发流程

Hibernate 应用程序的实现步骤：在了解了 Hibernate 的配置文件和映射文件的基础上，假设我们已经完成了配置文件和映射文件的配置工作，在进行 Hibernate 应用编程的过程中，我们应该按照表 5-1 的顺序来使用 5.3.2 节介绍的各个主要类来实现 Hibernate 的持久化操作。

要使用 Hibernate 支持实现原型系统中的模型层部分，即集成 Hibernate 和 Struts 进行应用开发需要按照以下步骤来修改已有的 Struts 应用。

(1) 添加 Hibernate 支持。

表 5-1　Hibernate 执行持久化操作的步骤

步骤	操　　作	代　　码
1	读取并解析配置文件	Configuration conf=new Configuration(). configure()
2	读取并解析映射文件，创建 SessionFactory	SessionFactory sf=conf. buildSessionFactory()
3	打开 Session	Session session=sf. openSession()
4	开始一个事务(增、删、改操作必须有事务支持，查询操作可选)	Transaction tx=session. beginTransaction()
5	持久化操作	session. save(user)　　//user 为已经实例化的对象
6	提交事务或回滚事务	tx. commit()(如果执行操作数据库的过程中出错，则应该执行回滚 tx. rollback())
7	关闭 Session	session. close()

(2) 使用 DataBase Explorer 视图创建映射文件与持久化类。

(3) 编写相应的 DAO 类来完成数据库访问操作的封装。

(4) 修改 Action，调用使用 Hibernate 支持完成数据库访问操作的函数。

> **提示：** Struts 和 Hibernate 集成开发的步骤可以灵活选择使用，例如，我们在开发第一个功能的时候已经添加了 Hibernate 支持，则可以直接从第二步开始。

后面的开发中我们将按照上述步骤来完成我们的原型系统改进工作。

5.4.2　Hibernate 开发原型中的注册功能

这里我们按功能要求，实现对原型系统中已有功能——注册——的模型层进行改进，使用 Hibernate 支持数据库的访问，并通过这个过程来演示如何开发一个 Hibernate 应用。

我们采用和前述应用相同的 MVC 结构。那么我们要修改的就只是原有应用的业务层和数据访问层的实现。

下面我们分步骤来构建这个 Hibernate 应用程序。具体的操作步骤如下：

(1) 为项目 strutsDemo 添加 Hibernate 支持。步骤如下列图示：

① 打开 DB Browser 视图：菜单 Window→show view→Other→MyEclipse DataBase→DB Browser(由于版本的不同，打开 DB Browser 视图的菜单路径会稍有不同)。

② 在 DB Browse 视图中右击，选择 New...菜单，如图 5-4 所示。



图 5-4　新建 database 连接驱动

③ 配置数据源各项参数：具体值如图 5-5 所示。

Database Driver
New Database Connection Driver
Create a new connection driver
Driver template: MySQL Connector/J 1
Driver name: mysql 2
Connection URL: jdbc:mysql://localhost:3306/mvcDB 3
User name: root 4
Password: **** 5
Driver JARs
E:\Project\workspace\strutsDemo\WebRoot\WEB-INF\lib\mysql-conn 6
Add JARs
Remove
Driver classname: com.mysql.jdbc.Driver
Test Driver
Connect to database on MyEclipse startup
Save password
Saved passwords are stored on your computer in a file that's difficult, but not impossible, for an intruder to read.
< Back Next > Finish Cancel

图 5-5 设置新数据连接驱动的各项参数

> **提示**：其中第 6 处的内容是通过 ADD JARs 按钮浏览找到的 JDBC 驱动文件。MySQL 的 JDBC 驱动包可以放在任何位置，只要能通过 ADD JARs 浏览找到即可。此处使用的驱动包是 3.11 版本。

④ 单击 Next 按钮，并在弹出的图中选择最后一个单选按钮，并单击 Add 按钮，进行数据库的选择，如图 5-6 和图 5-7 所示。



图 5-6 选择显示模式

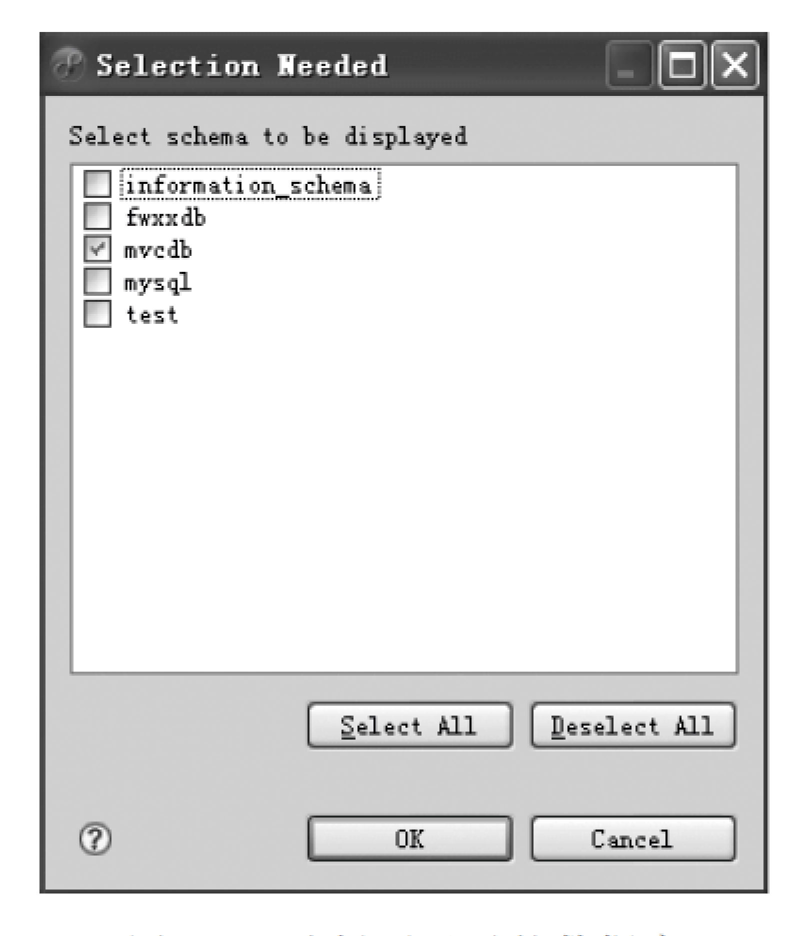


图 5-7 选择要显示的数据库

图 5-7 中显示的数据库列表会随着指定的数据库服务器和用户角色的不同而有所不同。

⑤ 选择我们需要的数据库并单击 OK 按钮返回，即可完成数据库驱动连接的配置。

> **提示：** 在添加 Hibernate 支持之前，如果已经完成了①～⑤步的操作，可以直接从第⑥步开始；①～⑤步的操作也可以在 Hibernate 支持添加完成之后，边界 Hibernate 配置文件的时候完成。

⑥ 右击项目 strutsDemo，并选择菜单：MyEclipse→add hibernate capabilities…。

⑦ 配置 Hibernate 基本信息：

在选择了第⑥步的菜单之后，会弹出图 5-8，此导航界面上要进行的配置项分别为：

- Hibernate Specification：选择要使用的 Hibernate 版本。
- Select the libraries to add to the buildpath：选择要使用的 Hibernate 库是来自 MyElipse 还是用户自定义库。
- JAR Library Istallation：要把 Hibernate 的 jar 文件添加到哪个项目位置。

各项的配置内容如图 5-8 所示。

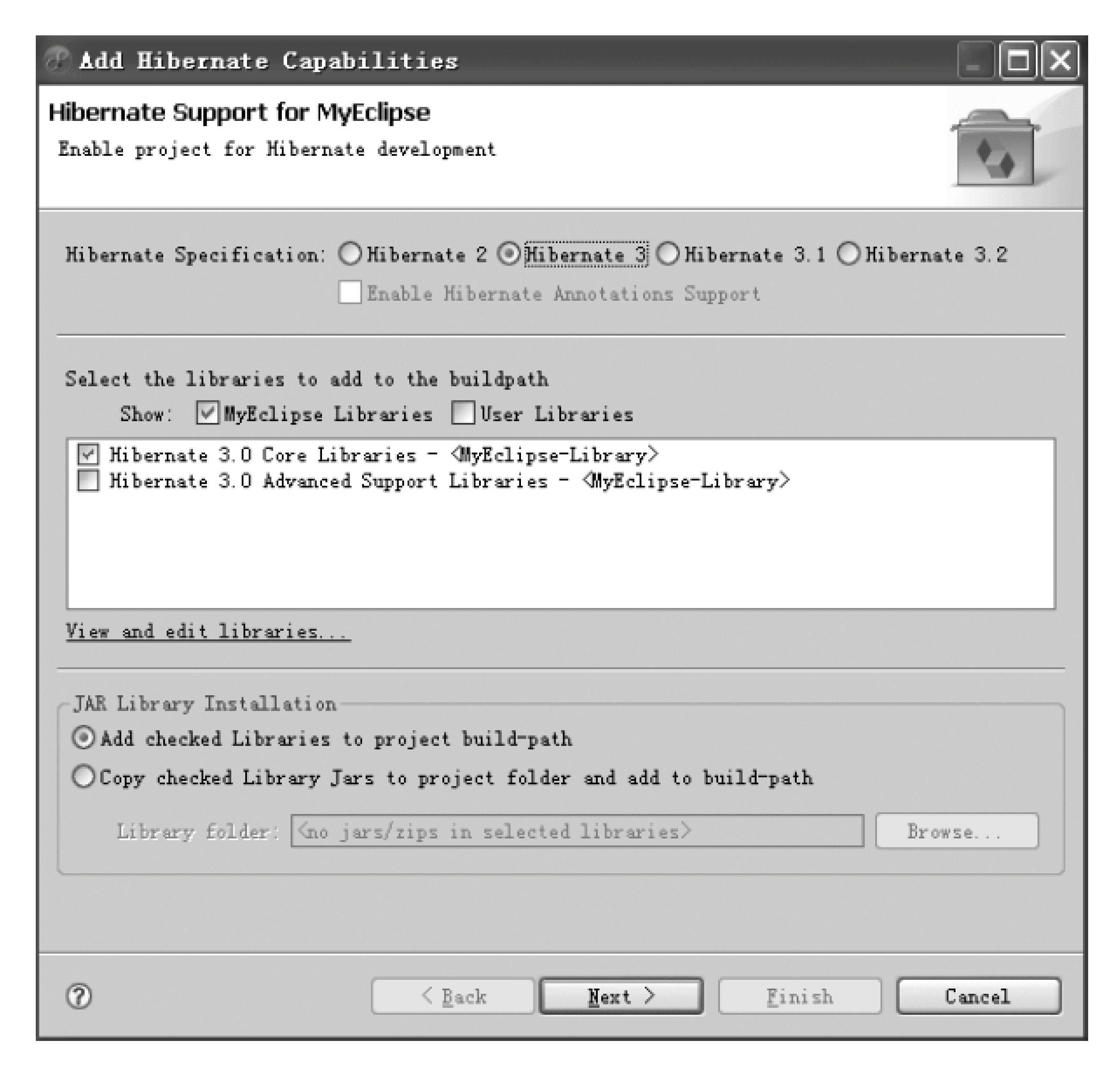


图 5-8 配置 Hibernate 基本信息

⑧ 创建 Hibernate 配置文件。单击图 5-8 中的 Next 按钮，就会弹出导航界面，进行 Hibernate 配置文件的创建界面如图 5-9 所示。这里需要配置的项有：

- Hibernate Config file：是新建一个配置文件还是使用已有的。
- Configuration Folder：配置文件存放的位置。
- Configuration File Name：配置文件的名字。默认名字是 hibernate. cfg. xml。这

里的名字是可以更改的。

✍ 最后一个复选框是问是否在向导结束的时候自动打开 hibernate. cfg. xml。

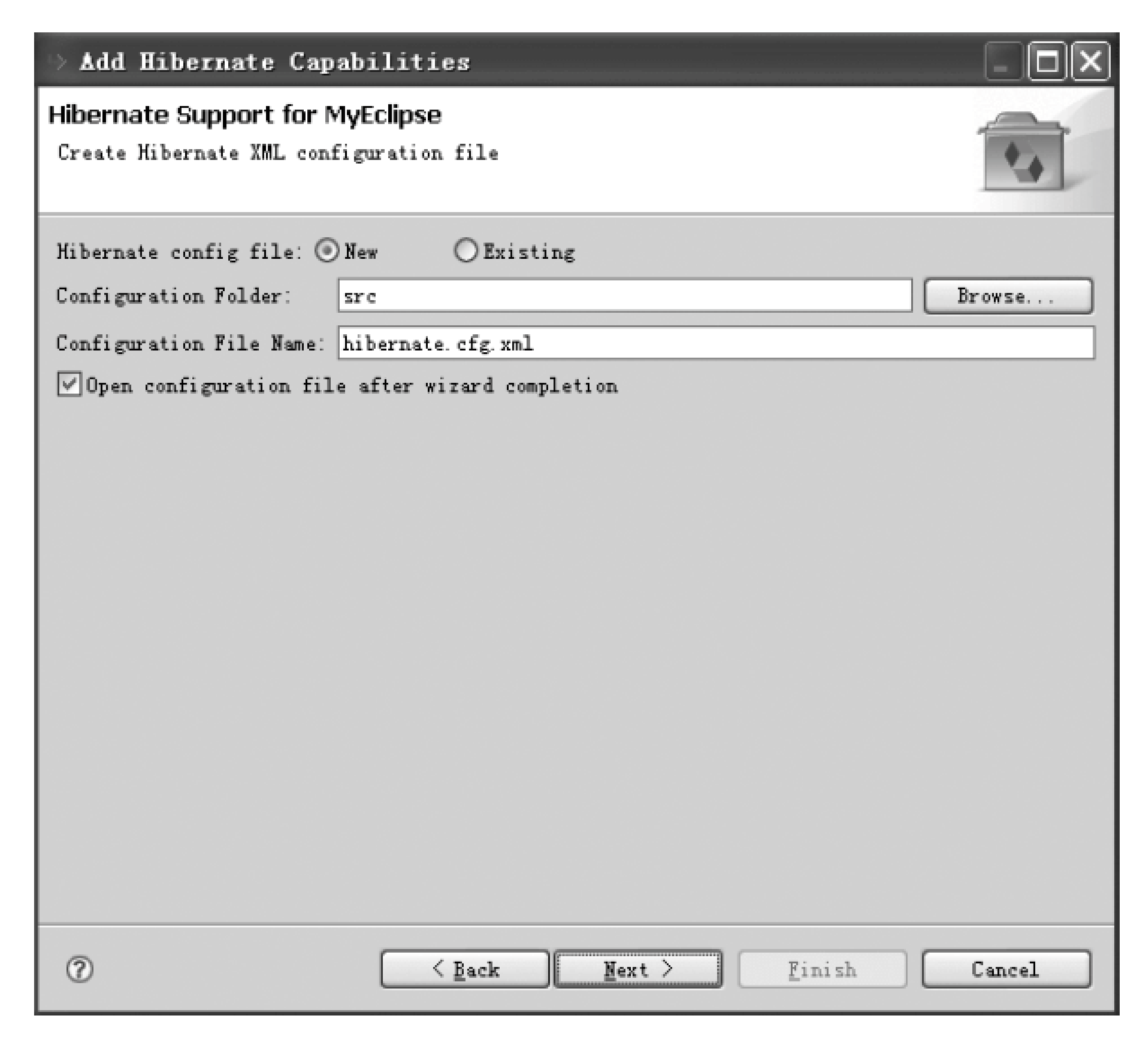


图 5-9　创建 Hibernate 配置文件

⑨ 配置 Hibernate 数据库连接的详细信息。单击图 5-9 中的 Next 按钮，弹出界面如图 5-10 所示，进行数据库连接驱动的指定。如果已经完成了第①～⑤步的操作，那么在下拉框中选择刚才在 DB Browser 中配置的数据源即可。之前没有操作第①～⑤步的配置，直接从第⑥步开始，则这里把第一个复选框去掉，继续进行下一步操作。



图 5-10　指定 Hibernate 数据库连接的细节

⑩ 定义 SessionFactory 的属性。这个向导界面引导程序员进行默认的 SessionFactory 类的相关设置，如图 5-11 所示。主要内容有：

✍ Create SessionFactory class：是否创建 SessionFactory 类。如果选中该项，表示要创建该类，那么要进行下列几项的配置，否则下列几项不可修改。

✍ Java Source Folder：此处选择工程中的 hibernate. cfg. xml 文件所在的位置。如果指定的文件夹中没有配置文件，则会报错。

✍ Java Package：这里指定要创建的 SessionFactory 类的位置。可以新建，也可以选择已有包。本项目中新建了一个包 com. wangyingling. hibernate. commons。

✍ ClassName：指定创建的 SessionFactory 类的类名。默认为 HibernateSessionFacttory。

指定默认生成文件 HibernateSessionFactory 类的默认包位置，如图 5-11 所示。

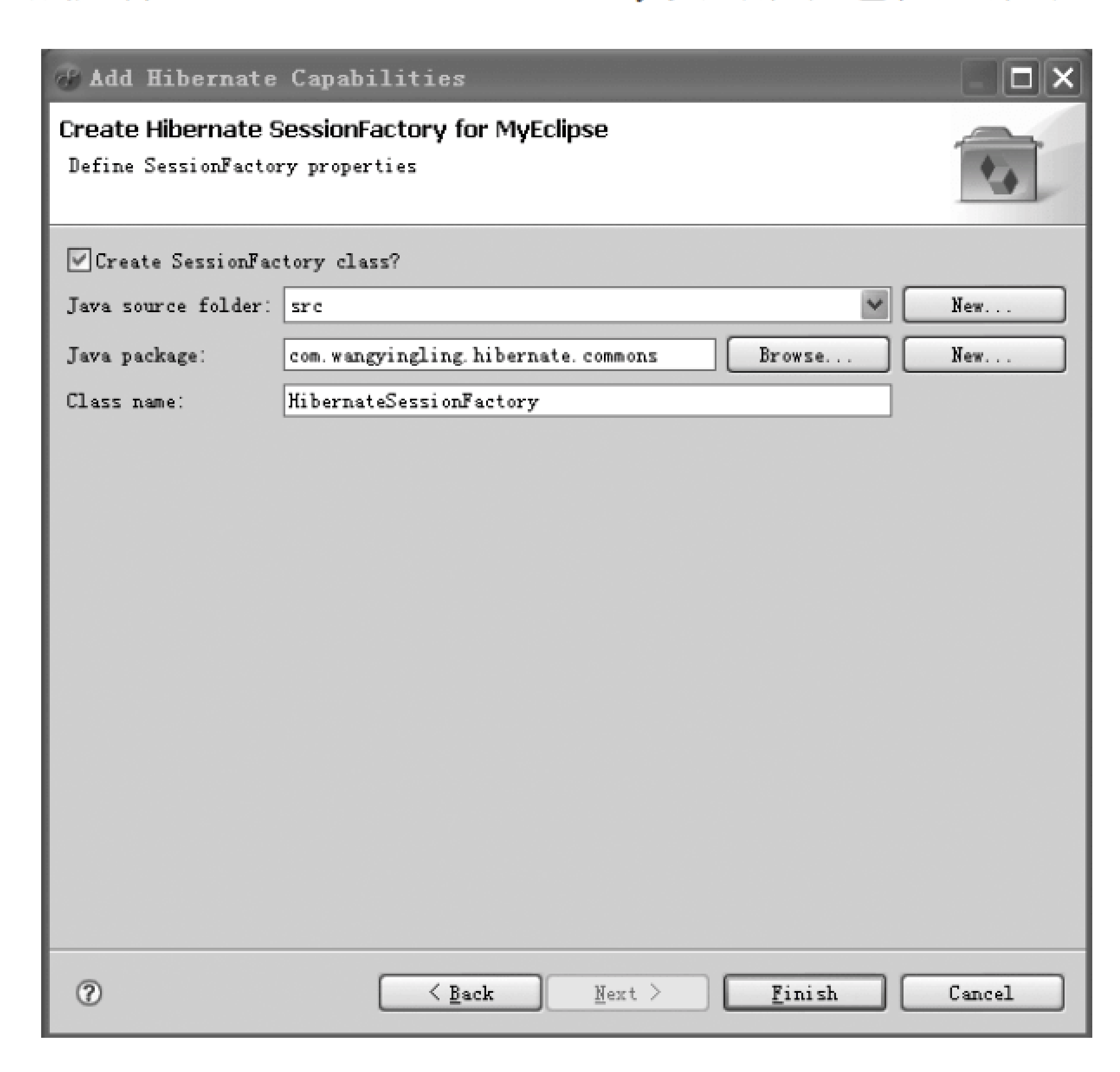


图 5-11 定义 SessionFactory 类

⑪ 单击 Finish 按钮完成 Hibernate 环境的配置。

⑫ 完成配置之后，配置文件的源代码如代码清单 5-4，HibernateSessionFactory 类的代码如代码清单 5-5。

代码清单 5-4 hibernate. cfg. xml 文件

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                   -->
<hibernate-configuration>
```

```
<session-factory>
    <property name="myeclipse.connection.profile">mysql</property>
    <property name="connection.url">
        jdbc:mysql://localhost:3306/mvcDB
    </property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <property name="connection.driver_class">
        com.mysql.jdbc.Driver
    </property>
    <property name="dialect">
        org.hibernate.dialect.MySQLDialect
    </property>

</session-factory>
</hibernate-configuration>
```

代码清单 5-5 HibernateSessionFactory.java

```
package com.wangyingling.hibernate.commons;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.cfg.Configuration;

/**
 * Configures and provides access to Hibernate sessions, tied to the
 * current thread of execution.  Follows the Thread Local Session
 * pattern, see {@link http://hibernate.org/42.html}.
 */
public class HibernateSessionFactory {

    /**
     * Location of hibernate.cfg.xml file.
     * Location should be on the classpath as Hibernate uses
     * #resourceAsStream style lookup for its configuration file.
     * The default classpath location of the hibernate config file is
     * in the default package. Use #setConfigFile() to update
     * the location of the configuration file for the current session.
     */
    private static String CONFIG_FILE_LOCATION="/hibernate.cfg.xml";
    private static final ThreadLocal threadLocal=new ThreadLocal();
    private  static Configuration configuration=new Configuration();
    private static org.hibernate.SessionFactory sessionFactory;
    private static String configFile=CONFIG_FILE_LOCATION;
```

```
static {
    try {
        configuration.configure(configFile);
        sessionFactory= configuration.buildSessionFactory();
    } catch (Exception e) {
        System.err
                .println("%%%% Error Creating SessionFactory %%%%");
        e.printStackTrace();
    }
}
private HibernateSessionFactory() {
}

/**
 * Returns the ThreadLocal Session instance.  Lazy initialize
 * the<code>SessionFactory</code> if needed.
 *
 * @return Session
 * @throws HibernateException
 */
public static Session getSession() throws HibernateException {
    Session session= (Session) threadLocal.get();

    if (session==null || !session.isOpen()) {
        if (sessionFactory==null) {
            rebuildSessionFactory();
        }
        session= (sessionFactory != null) ? sessionFactory.openSession()
                : null;
        threadLocal.set(session);
    }

    return session;
}

/**
 *  Rebuild hibernate session factory
 *
 */
public static void rebuildSessionFactory() {
    try {
        configuration.configure(configFile);
        sessionFactory= configuration.buildSessionFactory();
```

```
        } catch (Exception e) {
            System.err
                    .println("%%%% Error Creating SessionFactory %%%%");
            e.printStackTrace();
        }
    }

    /**
     * Close the single hibernate session instance.
     *
     * @throws HibernateException
     */
    public static void closeSession() throws HibernateException {
        Session session= (Session) threadLocal.get();
        threadLocal.set(null);

        if (session != null) {
            session.close();
        }
    }

    /**
     * return session factory
     *
     */
    public static org.hibernate.SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    /**
     * return session factory
     *
     * session factory will be rebuilded in the next call
     */
    public static void setConfigFile(String configFile) {
        HibernateSessionFactory.configFile= configFile;
        sessionFactory= null;
    }

    /**
     * return hibernate configuration
     *
     */
    public static Configuration getConfiguration() {
```

```
            return configuration;
        }

    }
```

(2) 创建映射文件与持久化类。

① 打开 DB Browser 中的数据源 mysql，找到 TABLE 中的 TblUser 表。选中 mysql 数据源，右击菜单 Open Connection..就可以打开连接(如果需要输入数据库用户密码请输入 root)。连接后的数据库视图如图 5-12 所示。

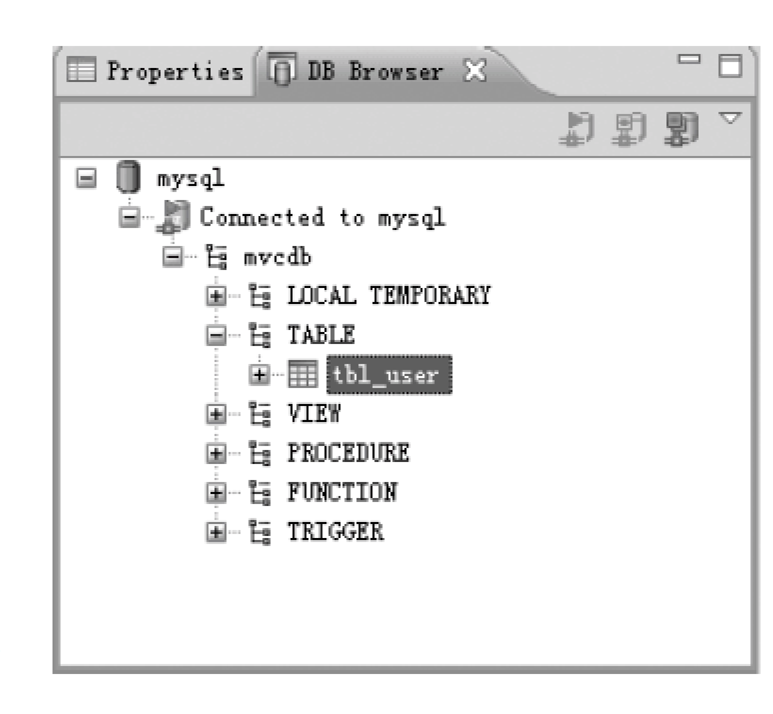


图 5-12　连接后的 DB Browser 视图

② 右击 tbl_user 表，选择倒数第二个弹出菜单项：Reverse Hibernate Enginerring。弹出如图 5-13 所示的导航界面。

✍ Java src folder：选择要使用该映射的工程的 src 目录，即 Hibernate 配置文件所在的目录。

✍ Java package：要输入生成的映射文件及对应的实体类所存放的包名，如果指定的包名不存在，导航在创建类的同时创建包。

✍ 其余处请按图示进行操作。

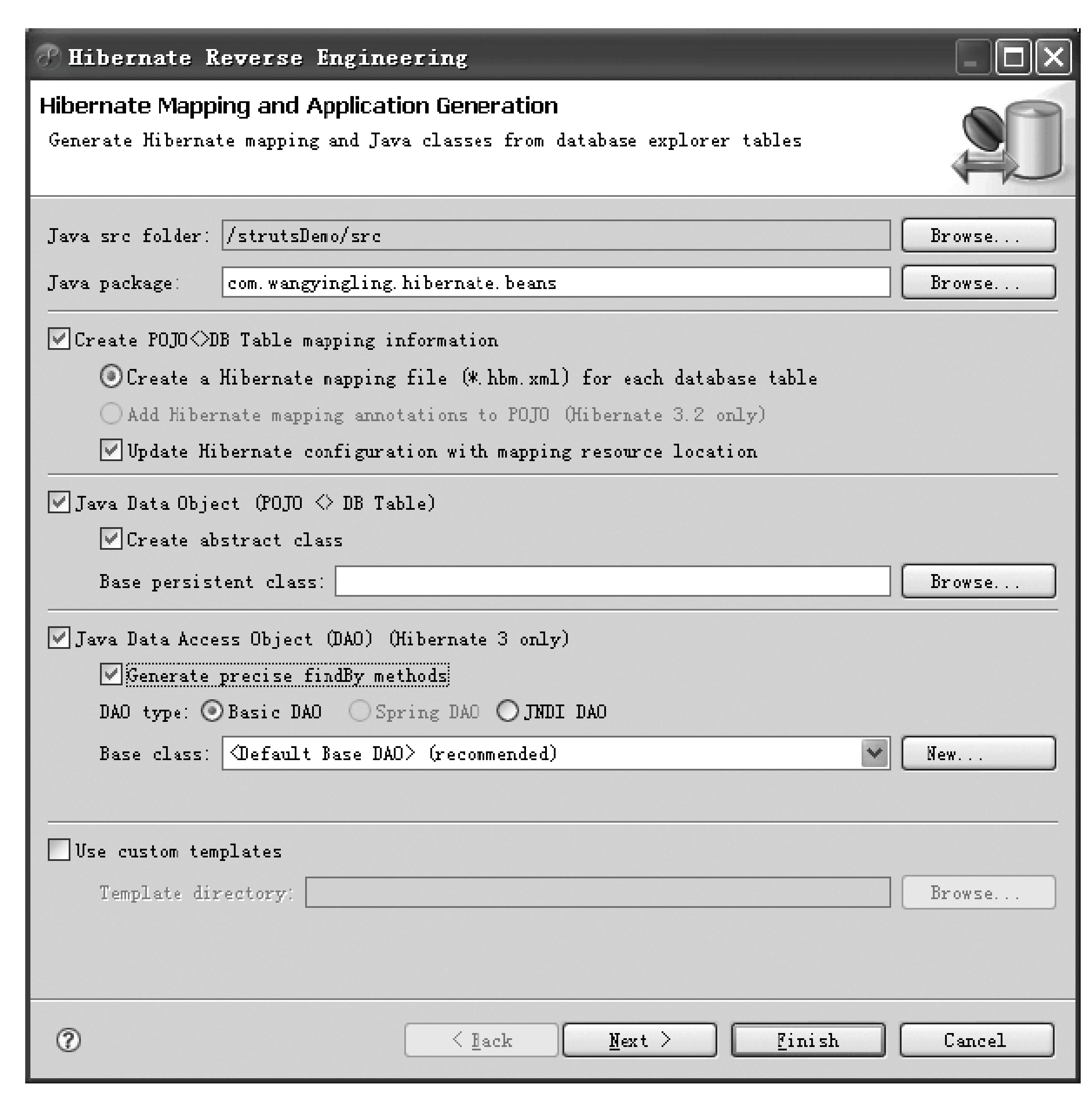


图 5-13　生成映射文件并从表生成对应的实体类和 DAO 类

③ 配置映射文件中类型映射的详细信息。单击图 5-13 中的 Next 按钮，出现图 5-14 所示的导航界面。需要进行如图 5-14 所示的操作。

✍ Id Generator：指定主键的生成方式。

✍ Customized Type Mappings：指定 Java 中的类型和表中的列类型之间的映射关系。如果不指定，则一切采取默认映射。

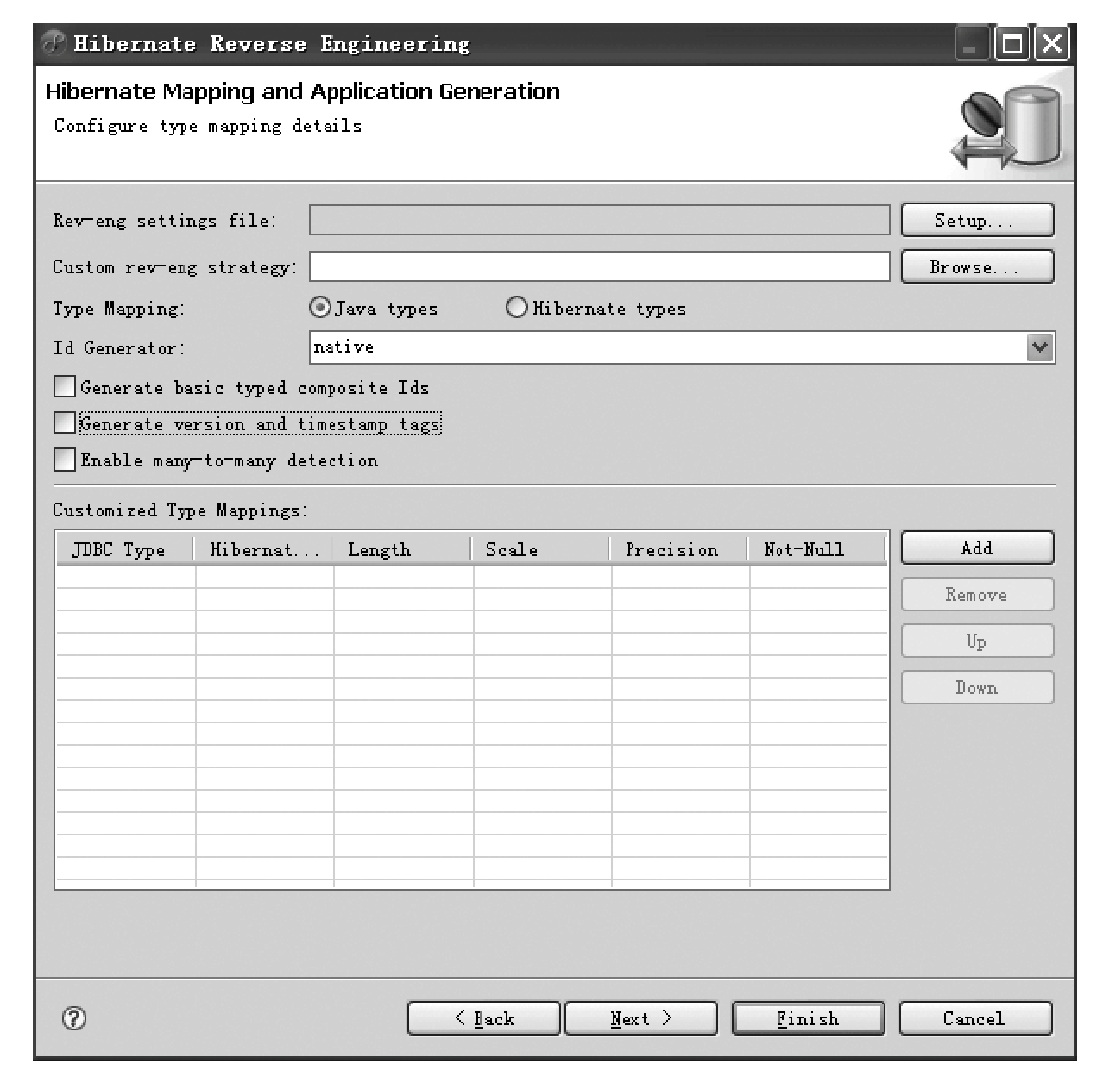


图 5-14　配置类型映射详细信息

④ 其余步骤都选择默认方式进行，一直到配置映射结束。所有的配置工作完成。

⑤ 完成映射之后生成的程序结构如图 5-15 所示，映射文件的内容如代码清单 5-6。

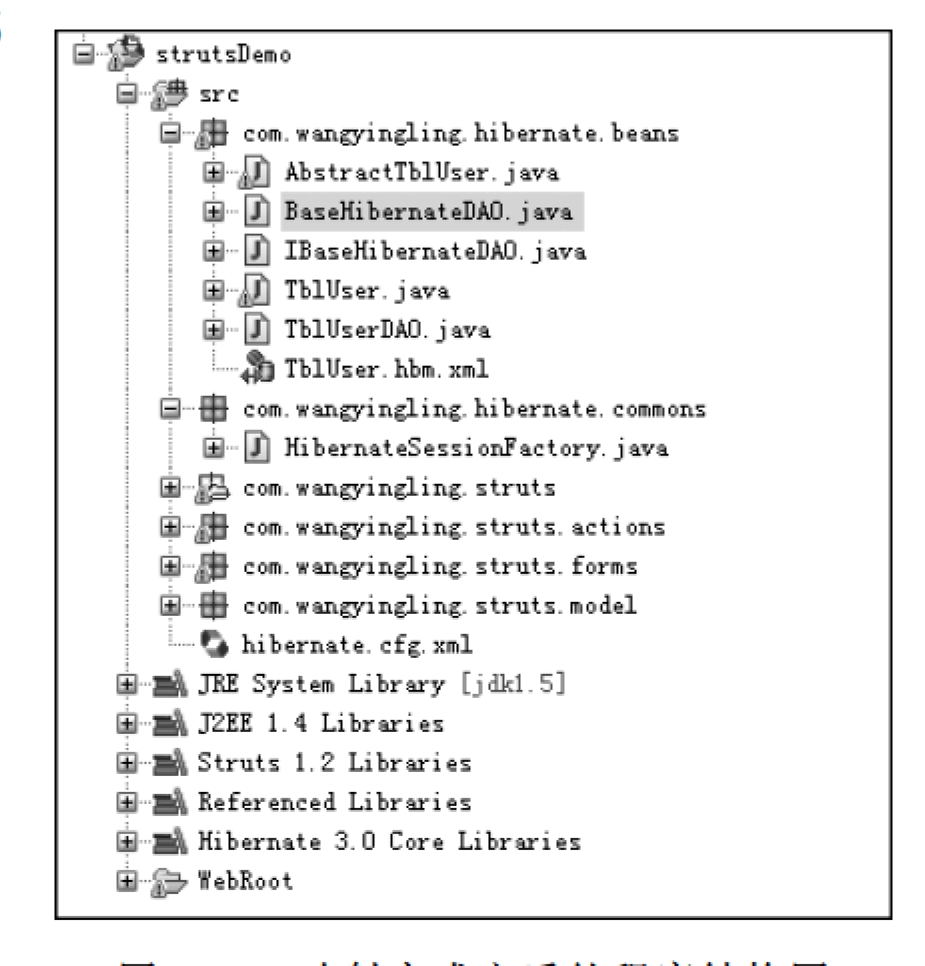


图 5-15　映射完成之后的程序结构图

代码清单 5-6　TblUser.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0
//EN"
"http://hibernate.sourceforge.net/
hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by
    MyEclipse Persistence Tools
-->
<hibernate-mapping>
    <class name="com.wangyingling
    .hibernate.beans.TblUser" table="tbl_user" catalog="mvcdb">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="native" />
```

```
        </id>
        <property name="uname" type="java.lang.String">
            <column name="uname" length="50" not-null="true" />
        </property>
        <property name="upass" type="java.lang.String">
            <column name="upass" length="20" not-null="true" />
        </property>
        <property name="email" type="java.lang.String">
            <column name="email" length="50" />
        </property>
    </class>
</hibernate-mapping>
```

(3) 编写 DAO 类。

① BaseHibernateDAO 类的编写。该类作为所有 DAO 类的父类,负责处理一些公共的 Hibernate 操作,如获得和关闭 Session。修改后的代码如代码清单 5-7 所示,包含两个关键方法:

✍ 获得 Session 的方法——getSession()。

✍ 关闭 Session 的方法——closeSession()。

代码清单 5-7 BaseHibernateDAO

```
package com.wangyingling.hibernate.beans;

import com.wangyingling.hibernate.commons.HibernateSessionFactory;
import org.hibernate.Session;


/**
 * Data access object (DAO) for domain model
 * @author MyEclipse Persistence Tools
 */
public class BaseHibernateDAO implements IBaseHibernateDAO {

    public Session getSession(){
        return HibernateSessionFactory.getSession();
    }
    public void closeSession(){
        HibernateSessionFactory.closeSession();
    }

}
```

② TblUserDAO 类的编写。该类实现的功能主要有:

✍ 添加用户——add(String username,String password,String email)。在 TblUserDAO

类中添加如代码清单 5-8 所示的代码。

✍ 验证用户有效性和用户是否存在的方法需要查询数据库记录，所以我们放在下一章实现，这里只给出要实现的方法的轮廓，如代码清单 5-8。

代码清单 5-8 添加到 TblUserDAO 中的 add()方法

```
/*
 * 添加用户
 */
public boolean add(String username, String password, String email)
            throws HibernateException {
        Session session= null;
        Transaction tx= null;
        boolean b= true;
        try {
            session= this.getSession();
            tx= session.beginTransaction();

            TblUser user= new TblUser();
            user.setUname(username);
            user.setUpass(password);
            user.setEmail(email);

            session.save(user);
            tx.commit();
        } catch (HibernateException e) {
            b= false;
            throw e;
        } finally {
            if (tx != null) {
                tx.rollback();
            }
            this.closeSession();
        }
        return b;
            }
/*
 * 验证用户是否有效
 */
    public TblUser valid(String username, String password)
            throws HibernateException {
        return null;
        }
/*
 * 判断用户是否存在
 */
```

```
    public boolean isExist(String username) throws HibernateException {
return true;
}
```

(4) UserAction类的insert方法的修改。该方法调用模型层的方法实现用户注册功能,所以我们对其进行修改,使其调用TblUserDAO类的add()方法。修改后的方法代码如代码清单5-9所示。

代码清单5-9 修改后的UserAction类中的add()方法

```
private void insert(HttpServletRequest request, UserForm registerForm){
    //调用业务层方法实现注册
    TblUserDAO uhb=new TblUserDAO();
    uhb.add(registerForm.getUname(), registerForm.getUpass(),
    registerForm.getEmail());
}
```

(5) 测试运行。

① 重新发布strutsDemo工程并运行服务器。

② 在浏览器中输入地址:http://localhost:8080/strutsDemo/register.jsp。

③ 输入注册信息并提交,然后查看数据库中的数据,看是否正常运行。

5.5 实验与能力拓展

实验指导:

1. 目标:上机练习完成课堂的用户管理功能的MVC结构的持久化实现的修改。

2. 工具以及配置说明:MyEclipse 7.0、MySQL 5配合开发Web应用的具体操作请参考课本。

3. 推荐步骤:

a) 添加Hibernate支持

b) 添加数据源配置

c) 生成配置文件

d) 根据数据表生成映射文件以及PO类

e) 修改数据访问层的代码

f) 修改Action类对DAO类的调用

g) 运行Web程序

4. 运行说明:需要指定Web服务器为Tomcat。其运行步骤请参考上册课本。

能力拓展:

1. 根据用户登录功能的Struts和Hibernate实现,完成用户注册功能的Struts和Hibernate实现。

2. 假定我们的程序因为结构的修改需要,把用户表对应的PO类名修改了一个,那么我们要如何修改其他的各个配置文件,使得我们的程序依然能正确运行?

第 6 章　Hibernate 查询

本章导读

前面章节中我们介绍了 Struts 和 Hibernate 集成开发的流程，并进行了修改数据库的操作案例练习。本章将继续进行 Struts 和 Hibernate 集成开发，但是重点转移到使用 Hibernate 查询的使用上，包括 HQL 和 Criteria Query。

工作任务

集成 Hibernate 和 Struts 框架，实现原型中的登录功能。

6.1　Hibernate 查询介绍

数据查询与数据检索是 Hibernate 的亮点之一。Hibernate 的数据查询主要有如下三种：

(1) Hibernate Query Language(HQL)。

(2) Criteria Query。

(3) Native SQL。

下面我们主要讨论 HQL。

Hibernate Query Language 提供了十分强大的功能，推荐大家使用这种查询方式。HQL 具有与 SQL 语言类似的语法规范，只不过 SQL 是针对数据表中的字段进行查询，而 HQL 是针对持久化对象，它用来取得对象，而不进行 update、delete 和 insert 等操作。而且 HQL 是完全面向对象的，具备继承、多态和关联等特性。

执行 HQL 语句的应用程序从结构和步骤可以分为以下几步，如表 6-1 所示。

表 6-1　HQL 语句的应用程序结构

步骤说明	代 码 示 例
得到 session	Session session=super.getSession()
编写 HQL 语句	String hql="FROM TblUser"
创建 Query 对象	Query query=session.createQuery(hql)
执行查询得到结果	List list=query.list()

6.2 HQL 查询基础

6.2.1 from 子查询

from 子句是最简单的 HQL 语句，例如 from TblUser，也可以写为 select s from TblUser s。它简单地返回 Student 类的所有实例。

除了 Java 类和属性的名称外，HQL 语句对大小写并不敏感，所以在上一句 HQL 语句中，from 与 FROM 是相同的，但是 TblUser 与 tblUser 就不同了。所以上句中 from 后跟的单词是 Java 类名，如果大小写写错了，就会报错。我们应用表 6-1 的应用程序结构，可以写出 from 子句的简单应用程序代码如清单 6-1。

代码清单 6-1 from 子句程序示例

```
Query query= session.createQuery("from TblUser");
List list= query.list();
For(int i= 0;i< list.size();i+ + )
{
   TblUser stu= ( TblUser)list.get(i);
   System.out.println(stu.getUname());
}
```

6.2.2 select 子查询

有时候我们并不需要得到对象的所有属性，这时就可以使用 select 子句进行属性查询，例如：select s. uname from TblUser s。结合表 6-1，程序清单 6-2 说明了如何执行 select 子句。

代码清单 6-2 select 子句程序示例 1

```
Query query= session.createQuery("select s.uname from TblUser s");
List list= query.list();
For(int i= 0;i< list.size();i+ + )
{
   String name= (String)list.get(i);
   System.out.println(name);
}
```

如果要查询两个以上的属性，查询结果就会以数组的方式返回，如代码清单 6-3 所示。

代码清单 6-3 select 子句程序示例 2

```
Query query= session.createQuery("select s.name,s.sex from TblUser s");
List list= query.list();
For(int i= 0;i< list.size();i+ + )
{
```

```
    Object[] obj= (Object[])list.get(i);
    System.out.println(obj[0]+ "的性别是："+ obj[1]);
}
```

分析代码清单 6-3 可以发现，在使用属性查询时，由于查询结果使用对象数组，操作和理解都比较麻烦，如果将一个 Object[]数组中的所有成员封装成一个对象就方便多了。程序清单 6-4 将查询结果进行了优化。

代码清单 6-4 select 子句多属性查询优化

```
Query query= session.createQuery("select new TblUser(s.uname,s.upass)
from TblUser s");
List list= query.list();
For(int i= 0;i< list.size();i+ + )
{
  TblUser stu= ( TblUser)list.get(i);
   System.out.println(stu.getUname());
}
```

要使程序清单 6-4 能正确运行，还需要在 Student 类中加入一个如程序清单 6-5 的构造函数。

代码清单 6-5 select 子句多属性查询优化的对应 PO 类构造函数

```
public TblUser (String name,String pwd)
{
this.uname= name;
this.upass= pwd;
}
```

6.2.3 where 子查询

HQL 也支持子查询，它通过 where 子句实现这一机制。where 子句可以让用户缩小要返回的实例的列表范围。

where 子句允许出现的表达式包括了 SQL 中可以使用的大多数情况：

（1）数学操作：＋ － * /。

（2）真假比较操作：＝ ＞＝ ＜＝ !＝ like。

（3）逻辑操作符：and or not。

（4）字符串连接：‖ 。

（5）SQL 标量函数：例如 supper()和 lower()。

如果子查询返回多条记录，可以用以下的关键字来量化：

（1）all：表示所有记录。

（2）any：表示所有记录中的任意一条。

（3）some：与 any 用法相同。

（4）in：与 any 等价。

(5) exists：表示查询至少要返回一条记录。

例如下面的语句返回所有学生年龄都大于 22 的班级对象

```
from Classes g where 22< all (select s.age from g.student s)
```

下述语句返回在所有学生中有一个学生的年龄等于 22 的班级：

```
from Classes g where 22= any (select s.age from g.student s)
```

或者

```
from Classes g where 22= some (select s.age from g.student s)
```

或者

```
from Classes g where 22= in (select s.age from g.student s)
```

6.2.4 order by 子查询

查询返回的列表可以按照任何返回的类或者组件的属性排序：

```
from Student s order by s.name asc
```

asc 和 desc 是可选的，分别代表升序或降序。

6.2.5 统计函数查询

可以在 HQL 中使用函数，常用的函数如下：

(1) count()：统计记录条数。

(2) min()：求最小值。

(3) max()：求最大值。

(4) sum()：求和。

(5) age()：求平均值。

使用了函数的 HQL 语句的执行程序结构和处理都和前述应用示例相似，所以我们不再一一给出完成的程序片断，只对 HQL 语句分别说明如下。

如果要取得 Student 实例的数量，可以编写如下的 HQL 语句：

```
select count(* ) from Student
```

取得 Student 平均年龄的 HQL 语句如下：

```
select age(s.age) from Student as s
```

可以使用 distinct 去除重复数据：

```
select distinct s.age from Student as s
```

 **提示：** from 后的两个空格分开的类名和别名之间的 as 可有可无。

6.2.6　联接查询

与 SQL 查询一样，HQL 也支持连接查询，如内连接、外连接和交叉连接。语法关键字如下：

（1）inner join：内连接。

（2）left outer join：左外连接。

（3）right outer join：右外连接。

（4）full join：全连接。不常用。

我们这里重点介绍普通项目中使用最为频繁的内联接查询。

inner join 可以简写为 join，例如在查询得到 Group 对象时，内联接取得对应的 Student 对象，实现的程序代码如代码清单 6-6 所示。

代码清单 6-6　内联接查询代码片断

```
…
//打开 Session,开启事务
Student stu=null;                                   //声明 Student 实例
Group group=null;                                   //声明 Group 实例
Query query=session.createQuery("from Group g join g.students");
List list=query.list();
Object obj[]=null;                                  //声明对象数组
For(int i=0;i<list.size();i++){
  obj=(Object[])list.get(i);                        //取得集合中的第 i 个数组
  group=(Group)obj[0];                              //group 是数组中第一个对象
  stu=(Student)obj[1];                              //stu 是数组中第二个对象
  System.out.println(stu.getName()+"属于"+group.getName());
}
…
//提交事务,关闭 Session
```

6.3　Criteria 查询

6.3.1　Criteria 查询的使用步骤

使用 Criteria 查询支持动态地使用一种面向对象的 API 创建查询，而不在代码中嵌入字符串。使用 Criteria 查询使用步骤一般如下：

（1）创建 Criteria 实例。

（2）添加查询条件。

（3）执行查询。

下面各小节详细讲述 Criteria 查询的各个步骤。

6.3.2 创建 Criteria 查询

当查询数据时,人们往往需要设置查询条件。在 SQL 或 HQL 语句中,查询条件常常放在 where 子句中。此外,Hibernate 还支持 Criteria 查询,这种查询方式把查询条件封装为一个 Criteria 对象。在实际应用中,可以使用 Session 的 createCriteria()方法构建一个 org. hibernate. Criteria 实例,然后把具体的查询条件通过 Critetia 对象的 add()方法加入到 Criteria 实例中。这样,程序员可以在不适用 SQL 甚至 HQL 的情况下进行数据查询。

关于 Criteria 查询的简单使用,请参考代码清单 6-7。

代码清单 6-7 Criteria 查询代码片断

```
Criteria cr=session.createCriteria(TblUser.class);  //生成一个 Criteria 对象
cr.add(Restriction.eq("name","Bill"));              //等价于 where name='Bill'
List list=cr.list();
TblUser stu=(TblUser)list.get(0);
System.out.println(stu.getUname());
```

6.3.3 使用 Restriction 类为查询增加限制

在代码清单 6-7 中,Restrictions. eq()方法表示 equal,即等于的情况。Restrictions 类提供了查询限制机制。它提供了许多方法,以实现查询限制。这些方法和其他一些 Criteria Query 常用查询方法请参考表 6-2。

表 6-2 Criteria Query 常用查询方法

方　法	描　述
Restrictions. eq	=
Restrictions. allEq	利用 Map 来进行多个等于的限制
Restrictions. gt	>
Restrictions. ge	>=
Restrictions. lt	<
Restrictions. le	<=
Restrictions. between	BETWEEN
Restrictions. like	LIKE
Restrictions. in	IN
Restrictions. and	AND
Restrictions. or	OR
Restrictions. sqlRestriction	用 SQL 限定查询

下面,我们通过几个程序片断介绍如何使用 Criteria 类的常用方法。

- 查询用户名以 t 开头的所有 Student 对象。

```
Criteria cr=session.createCriteria(TblUser.class);          //创建 Criteria
```

```
cr.add(Restrictions.like("uname","t%"));                //添加查询条件
List list=cr.list();                                    //执行查询
TblUser stu=(TblUser)list.get(0);
```

或者使用另外一种方式：

```
Criteria cr=session.createCriteria(TblUser.class);      //创建 Criteria
cr.add(Restrictions.like("uname","t",MatchMode.START)); //添加查询条件
List list=cr.list();                                    //执行查询
TblUser stu=(TblUser)list.get(0);
```

- 查询用户姓名在 Bill、Jack 和 Tom 之间的所有 TblUser 对象。

```
String[] names={"Bill","Jack","Tom"};
Criteria cr=session.createCriteria(TblUser.class);      //创建 Criteria
cr.add(Restrictions.in("uname",names));                 //添加查询条件
List list=cr.list();                                    //执行查询
TblUser stu=(TblUser)list.get(0);
```

- 查询用户姓名以字母 F 开头的所有 TblUser 对象，并按照姓名升序排列。

```
Criteria cr=session.createCriteria(TblUser.class);      //创建 Criteria
cr.add(Restrictions.like("uname","F%"));                //添加查询条件
cr.addOrder(Order.asc("uname"));                        //添加查询条件
List list=cr.list();                                    //执行查询
TblUser stu=(TblUser)list.get(0);
```

关于查询的其他种类和方式，我们这里将不再一一介绍。如果感兴趣，同学们可以参考相关的参考资料和网站。

6.4 应用 HQL 查询实现原型系统的登录功能

实现登录功能的 Hibernate 改造的过程仍然是使用 Hibernate 开发模型层的过程，仍然是 Hibernate 和 Struts 集成的过程。所以我们采用的仍然是在 5.4.1 节介绍的过程。

因为在开发注册功能的过程中我们已经实现了 Hibernate 环境的添加和映射文件及实体类的实现，所以我们直接从第三步开始进行登录功能的实现。

(1) 修改 TblUserDAO 中的 valid()方法和 isExist()方法。修改后的代码如代码清单 6-8 所示。

代码清单 6-8 TblUserDAO 类中的 valid()方法和 isExist()方法

```
/*
 * 验证用户是否有效
 */
    public TblUser valid(String username, String password)
            throws HibernateException {
```

```
        Session session= null;
        Transaction tx= null;
        TblUser user= null;
        try {
            session= this.getSession();
            tx= session.beginTransaction();

            Query query= session
                    .createQuery("from TblUser where uname= ? and upass= ?");
            query.setString(0, username.trim());
            query.setString(1, password.trim());

            user= (TblUser) query.uniqueResult();

            query= null;

            tx.commit();
        } catch (HibernateException e) {
            throw e;
        } finally {
            if (tx != null) {
                tx.rollback();
            }
            this.closeSession();
        }
        return user;
    }
/*
 * 判断用户是否存在
 */
    public boolean isExist(String username) throws HibernateException {
        Session session= null;
        Transaction tx= null;
        boolean b= false;
        try {
            session= this.getSession();
            tx= session.beginTransaction();

            Query query= session.createQuery("from TblUser where uname= ?");
            query.setString(0, username.trim());

            TblUser user= (TblUser) query.uniqueResult();
            if (user != null)
                b= true;
```

```
            query=null;

            tx.commit();
        } catch (HibernateException e){
            throw e;
        } finally {
            if (tx !=null){
                tx.rollback();
            }
            this.closeSession();
        }
        return b;
    }
```

（2）修改 UserAction 类中的 valid()和 isExist()方法，修改后的代码如代码清单 6-9 所示。

代码清单 6-9　UserAction 类中的 valid()方法和 isExist()方法

```
private boolean isExist(HttpServletRequest request,
            UserForm registerForm){
        //调用业务层方法判断用户名是否已经存在
        TblUserDAO uhb=new TblUserDAO();
        boolean isExist=uhb.isExist(registerForm.getUname());
        return isExist;
    }

    private boolean valid(HttpServletRequest request, UserForm loginForm){
        TblUserDAO uhb=new TblUserDAO();
        boolean isValid=uhb.valid(loginForm.getUname(),
        loginForm.getUpass())==null?false:true;
        return isValid;
    }
```

（3）运行 Web 工程进行测试。

① 重新发布 strutsDemo 并运行服务器。

② 在 IE 地址栏中输入地址 http://localhost:8080/strutsDemo/login.jsp。

③ 按照程序流程进行测试即可。

6.5　实验与能力拓展

实验：

1. 按照教材内容进行查询的练习。

2. 请修改程序中的 TblUserDAO 类中的 isExist()方法和 valid()方法，使用 HQL 进行查询并处理查询结果，要求不修改 UserAction 类中的方法。

第 7 章 项目练手：网上购物子系统

本章导读

本章将提供一个小的通用网上购物子系统，作为大家的练手项目。您需要根据本章提供的演示 demo，采用 MVC 架构，使用 Struts 和 Hibernate 进行功能的实现。

通过本章的项目实战，您需要根据本书讲解过的各种技术和框架，熟练使用下述技术实现项目：

✍ 使用 Struts 框架处理 MVC 的视图层和控制层。

✍ 使用 Hibernate 处理 MVC 的模型层，进行持久化操作。

如果在本章项目的开发过程中有什么问题和技术难题，请参考前面章节的相关内容或参考其他相关资料。

工作任务

根据需求描述和演示 demo 实现网上购物子系统的用户管理功能，其他功能自己实现。具体要求请参考 7.1 节和 7.2 节。

7.1 系统概述

本书要实现的网购子系统主要有以下功能：

✍ 用户登录注册：用户通过用户名和密码可以登录系统，进入后台用户中心，进行信息的维护和管理。

✍ 个人信息查询和维护。

普通用户登录后，即可浏览自己的信息。其中包括用户登录时间、用户注册信息以及用户的账户订单概况。

✍ 订单查询。

用户可以根据订单号、订单类型和订单生成日期作为查询条件进行查询。

✍ 充值记录查询。

用户可以根据充值类型和生成日期作为查询条件进行查询。

7.2　系统功能演示

（1）如果用户是第一次进入本购物网站，进行购物之前需要进入注册页面，注册页面需要的信息如图 7-1 所示。

（2）如果用户输入的注册信息不符合要求，则会重新进入注册页面，并弹出对应的错误提示框进行警告。如果用户输入的注册信息无误，则会提示正确注册的信息，并转入登录页面。登录页面需要的信息如图 7-2 所示。



图 7-1　注册页面



图 7-2　网购子系统登录页面

（3）如果输入的信息有误，将弹出错误提示信息，并回到登录页面。如果输入的登录信息无误，将直接进入用户中心首页。用户中心首页界面如图 7-3 所示。

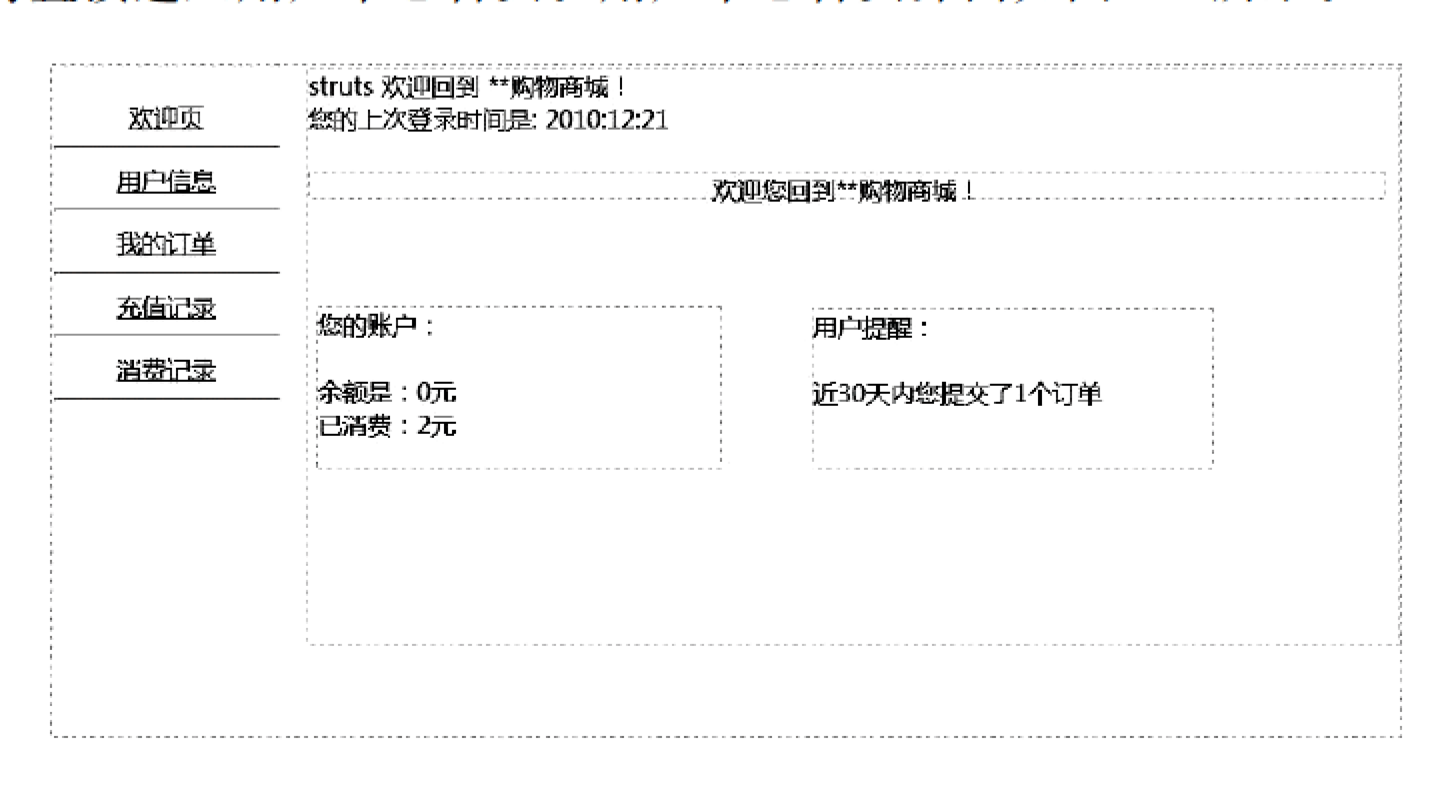


图 7-3　网购子系统用户中心首页

（4）单击左侧导航栏的“我的订单”可以进入到订单查询页面。订单查询页面的显示信息和查询条件信息如图 7-4 所示。

（5）用户信息的查询和修改页面内容如图 7-5 所示。其中用户名不可修改。



图 7-4　网购子系统的订单查询页面

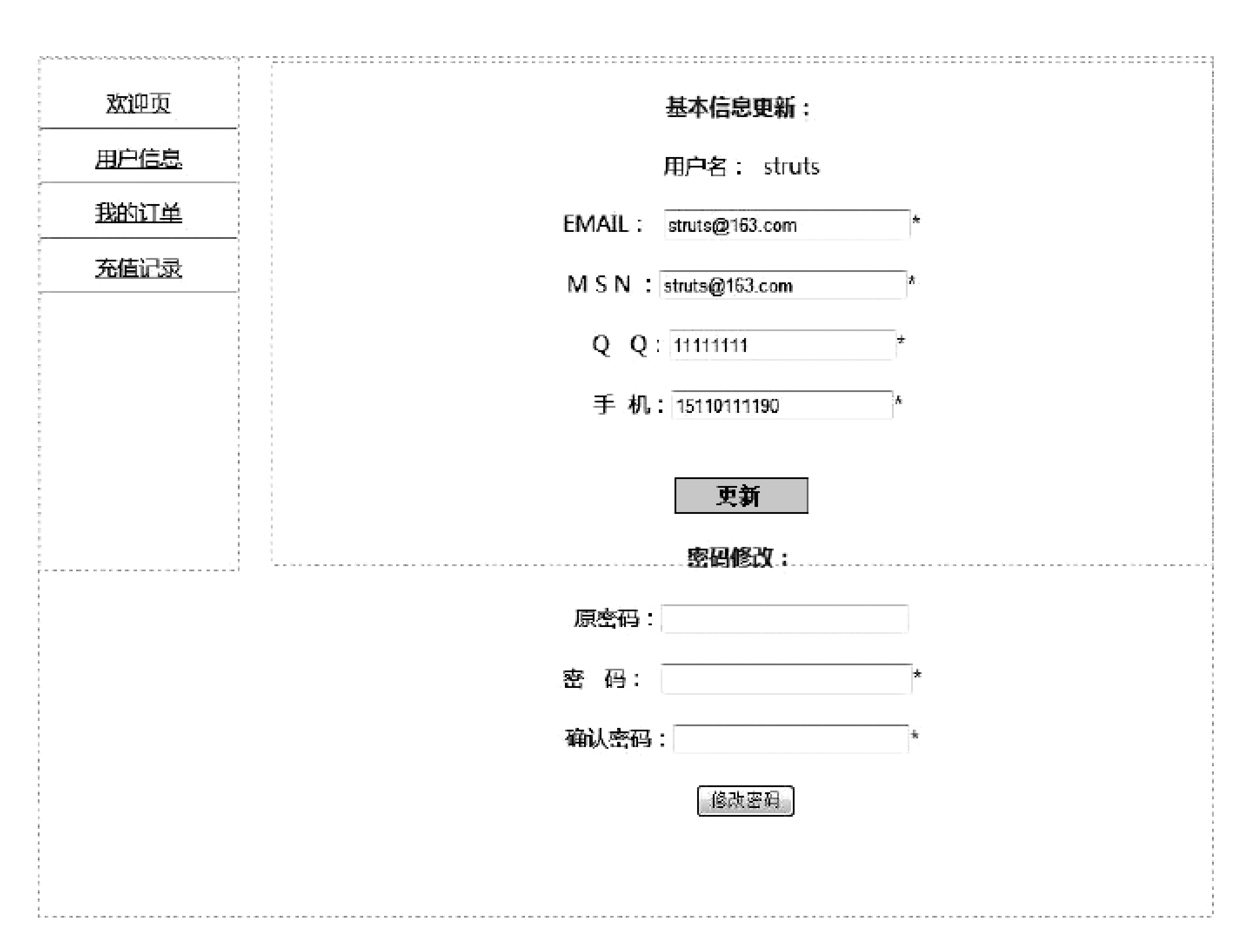


图 7-5　网购子系统的用户信息查询和管理

充值记录查询页面的显示信息和查询条件如图 7-6 所示页面。

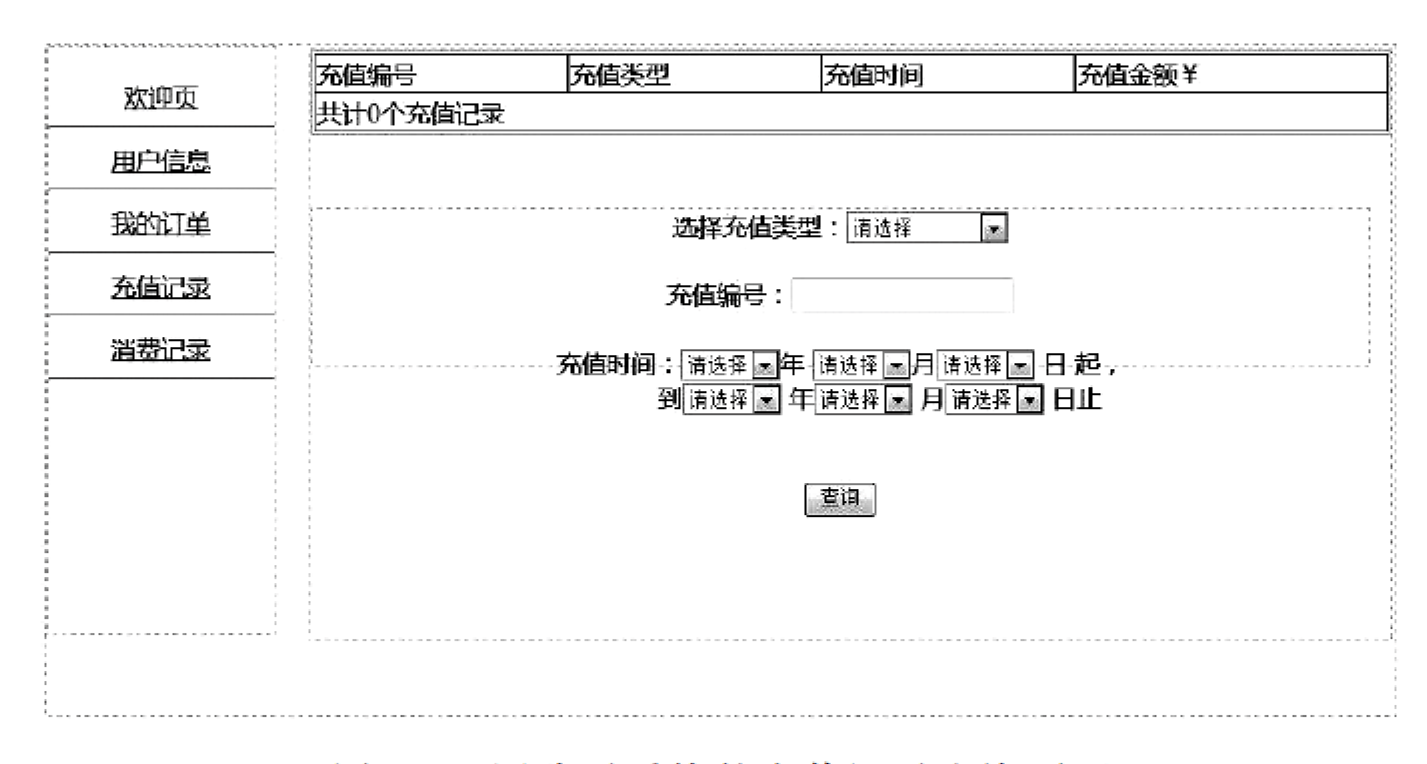


图 7-6　网购子系统的充值记录查询页面

7.3 系统设计

7.3.1 数据库设计

数据库的设计是系统设计中的重要环节，数据库的设计是在分析需求的基础上，结合数据库设计规范进行的。本项目中需要处理的主要实体有：用户、订单、充值记录。这些实体的详细信息如下。

（1）用户实体：用户名，密码，EMAIL，MSN，QQ，手机号码，登录时间，账户余额，已经消费金额。

（2）订单实体：订单编号，下单时间，交易金额，用户编号，产品名称，订单状态。

（3）充值记录：充值时间，充值金额，充值说明，用户编号，充值类型。

更多的实体分析我们这里不做考虑，仅以能实现本书的该项目为目的。根据上述分析，我们可以进行数据库物理表结构的设计。

7.3.2 创建数据库

本系统需要用到的数据库表结构如表 7-1～表 7-3 所示。

表 7-1 user_tbl（用户）表结构

列名	类型	允许空	备　注
id	int	不允许	主键，用户 ID
name	varchar(100)	不允许	用户名
password	varchar(10)	不允许	密码
EMAIL	varchar(200)	不允许	电子邮件
MSN	varchar(200)	不允许	MSN
Qq	varchar(20)	不允许	QQ
Tel	varchar(20)	不允许	电话
Logtime	Timestamp	不允许	登录时间
coin	int	允许	电子货币余额
consume	int	允许	消费的电子货币

表 7-2 order_tbl（订单）表结构

列　名	类　型	允　许　空	备　注
orderid	int	不允许	主键，订单编号
price	int	允许	订单金额
optime	timestamp	允许	添加时间
userid	int	不允许	外键，用户 ID
productname	varchar(500)	允许	商品名称
State	varchar(100)	不允许，状态为“已完成”、“等待付款”或“已经发货”	订单状态

表 7-3 add_tbl(充值记录)表结构

列名	类型	允许空	备　注
addid	int	不允许	主键,充值记录 id
type	varchar(50)	允许	充值类型
coin	int	允许	充值金额
datetime	datetime	不允许	充值日期
userid	int	允许	外键,用户 id

创建数据库的脚本代码如代码清单 7-1 所示。

代码清单 7-1 数据库的脚本代码

```
/* 创建数据库 */
CREATE DATABASE buy;
/* 使用数据库 */
USE buy;
/* 创建用户表 */
CREATE TABLE user_tbl(
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
NAME VARCHAR(100) NOT NULL,
PASSWORD VARCHAR(100) NOT NULL,
email VARCHAR(200) NOT NULL,
msn VARCHAR(200) NOT NULL,
qq VARCHAR(20) NOT NULL,
tel VARCHAR(20) NOT NULL,
logtime TIMESTAMP NOT NULL,
coin INT,
consume INT
);
/* 创建订单表 */
CREATE TABLE order_tbl(
orderid INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
price INT,
optime TIMESTAMP,
userid INT NOT NULL,
productname VARCHAR(500) NOT NULL,
state VARCHAR(100)

);
/* 创建充值记录表 */
CREATE TABLE add_tbl(
addid INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
TYPE VARCHAR(50),
coin INT,
optime TIMESTAMP NOT NULL,
```

```
userid INT
);

/* 插入测试数据 */
INSERT INTO 'user_tbl'('id','name','password','email','msn','qq','logtime',
'coin','consume')
VALUES (1,'first','first','first@163.com','first@163.com','123456',
'2010-01-01 00:00:00',0,0),
(2,'second','second','second@163.com','second@163.com','23456',
'2000-09-09 09:10:11',1,2);

INSERT INTO 'order_tbl'('orderid','price','optime','userid','productname','state')
VALUES (1,1,'2009-09-09 09:09:09',2,'故事会','完成'),
(2,1,'2008-08-02 02:09:09',2,'嘟嘟熊画报','完成');

INSERT  INTO 'add_tbl'('addid','type','coin','optime','userid')
VALUES (1,'网上银行转账',1,'2010-09-09 00:00:00',2);
```

7.3.3　目录和包结构

在进行开发之前，为了协作和维护方便，需要定义良好的目录和包结构。良好的项目结构都有共同点：逻辑清楚。本系统的源代码结构和 Web 目录结构分别如图 7-7 和图 7-8 所示。

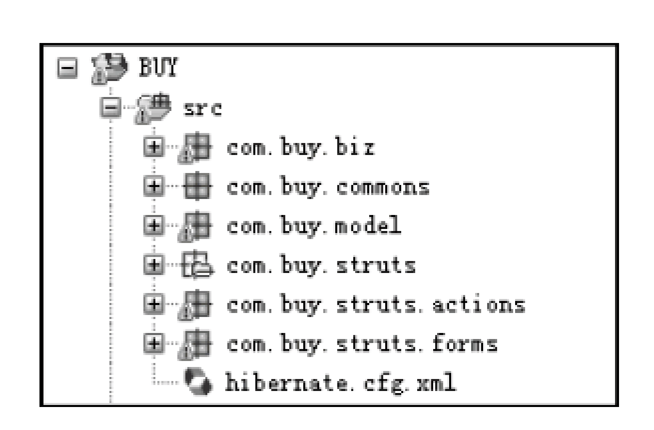


图 7-7　源代码包结构

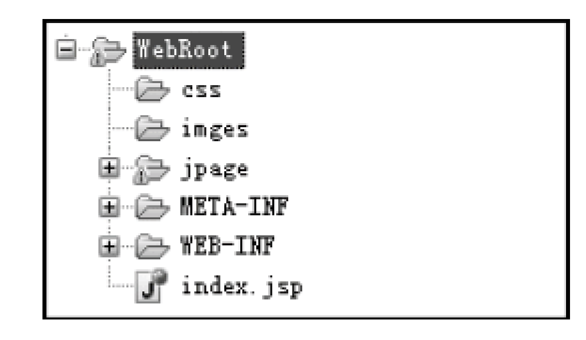


图 7-8　Web 目录结构

其中 BUY 是项目名称，所有的 Java 类都处在 src 文件夹下的不同包中。WebRoot 目录中存放网站的相关内容，WEB-INF 目录存放系统的配置文件。

根据开发中设计的不同要求，目录结构可以不同于图 7-8 的目录结构，只要结构便于维护，逻辑清晰即可。

7.4　工程准备

本系统开发过程中所采用的开发环境由如下开发工具组成：

✍ Web 服务器 Tomcat6.x。

✍ 数据库服务器 MySQL5.0.18。

✍ 开发平台为集成的 MyEclipse 7.0。
✍ 数据库客户端为 SQLyogv8.1.4。

7.5 工程的设计和实现

7.5.1 配置工程的 Struts 和 Hibernate 框架支持

这里我们将开始网购子系统的设计和实现。因为本工程规模很小，所以我们采用灵活的边设计边实现的方式进行工程的开发。本节，我们将创建工程并为创建的 Web 工程配置 Struts 和 Hibernate 框架的支持。

(1) 创建 Web 工程 BUY。

(2) 为工程添加 Struts1.2 支持。详细步骤请参考章节 2.1.1。

(3) 为工程添加 Hibernate3 支持。详细步骤请参考章节 5.4.2。

7.5.2 为工程添加公共类

本系统采用 Strust 和 Hibernate 架构进行开发，由 Hibernate 进行数据对象的操作和持久化，7.5.1 节配置 Hibernate 框架时自动生成的 com.buy.model.HibernateSessionFactory.java 类来负责全局 SessionFactory 实例的创建，并提供 Session 实例的打开和关闭等操作，即 com.buy.model.HibernateSessionFactory.java 类负责所有 Hibernate 的初始化操作，其代码如代码清单 7-2。

com.buy.model.HibernateSessionFactory.java 类的主要功能如下：

✍ 通过静态代码块实现 SessionFactory 的创建。
✍ getSession()：获得打开的 Session 对象。
✍ closeSession()：关闭 Session 对象。

代码清单 7-2 HibernateSessionFactory.java

```
package com.buy.model;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class HibernateSessionFactory {

    private static String CONFIG_FILE_LOCATION= "/hibernate.cfg.xml";
    private static final ThreadLocal< Session> threadLocal=
    new ThreadLocal< Session> ();           //该线程负责处理 session

    private   static Configuration configuration= new Configuration();
    private static org.hibernate.SessionFactory sessionFactory;
```

```
private static String configFile= CONFIG_FILE_LOCATION;

/**
 * 该静态代码块实现 SessionFactory 的创建
 */
static {
    try {
        configuration.configure(configFile);
        sessionFactory= configuration.buildSessionFactory();
    } catch (Exception e) {
        System.err
                .println("%%%% Error Creating SessionFactory %%%%");
        e.printStackTrace();
    }
}
private HibernateSessionFactory() {
}

/**
 * 获得打开的 Session 对象：Lazy initialize
 * the<code>SessionFactory</code> if needed.
 *
 * @return Session
 * @throws HibernateException
 */
public static Session getSession() throws HibernateException {
    Session session= (Session) threadLocal.get();

    if (session==null || !session.isOpen()) {
        if (sessionFactory==null) {
            rebuildSessionFactory();
        }
        session= (sessionFactory != null) ? sessionFactory.openSession()
                : null;
        threadLocal.set(session);
    }

    return session;
}

/**
 * Rebuild hibernate session factory
 *
 */
```

```
public static void rebuildSessionFactory() {
    try {
        configuration.configure(configFile);
        sessionFactory= configuration.buildSessionFactory();
    } catch (Exception e) {
        System.err
                .println("%%%% Error Creating SessionFactory %%%%");
        e.printStackTrace();
    }
}

/**
 * 关闭当前正在打开的 Session 对象
 *
 * @throws HibernateException
 */
public static void closeSession() throws HibernateException {
    Session session= (Session) threadLocal.get();
    threadLocal.set(null);

    if (session != null) {
        session.close();
    }
}

/**
 * return session factory
 *
 */
public static org.hibernate.SessionFactory getSessionFactory() {
    return sessionFactory;
}

/**
 * return session factory
 *
 * session factory will be rebuilded in the next call
 */
public static void setConfigFile(String configFile) {
    HibernateSessionFactory.configFile= configFile;
    sessionFactory= null;
}

/**
 * return hibernate configuration
 *
```

```
 */
public static Configuration getConfiguration(){
    return configuration;
}

}
```

7.5.3　实现 DAO 模式的公共类

创建一个 com.buy.model.BaseHibernateDAO.java 类，该类作为所有 DAO 类的父类，负责处理一些公共的 Hibernate 操作，如获得和关闭 Session。修改后的代码如代码清单 7-3 所示，包含两个关键方法：

✍ 获得 Session 的方法——getSession()。

✍ 关闭 Session 的方法——closeSession()。

代码清单 7-3　BaseHibernateDAO.java

```
package com.wangyingling.hibernate.beans;

import com.wangyingling.hibernate.commons.HibernateSessionFactory;
import org.hibernate.Session;


/**
 * Data access object (DAO) for domain model
 * @author MyEclipse Persistence Tools
 */
public class BaseHibernateDAO implements IBaseHibernateDAO {

    public Session getSession(){
        return HibernateSessionFactory.getSession();
    }
    public void closeSession(){
        HibernateSessionFactory.closeSession();
    }

}
```

7.6　用户管理功能的设计和实现

7.6.1　用户管理功能的逻辑设计

用户管理分为用户注册、用户登录和用户信息维护三个功能模块（图 7-9）。

- 用户登录功能主要流程是：从 login.jsp 进行登录，请求提交给 UserAction，如果

登录成功，则转到用户信息维护页面 userinfo. jsp；如果登录失败则转回到登录页面 login. jsp。

- 用户注册功能主要流程是：从 register. jsp 页面进行注册，注册请求提交给 UserAciton，如果注册成功，则转到登录页面 login. jsp，如果注册失败则转回到注册页面 register. jsp。
- 用户信息维护功能主要流程是：在 userinfo. jsp 页面进行信息的查看和修改，修改之后转回到 userinfo. jsp 页面，显示修改后的信息。

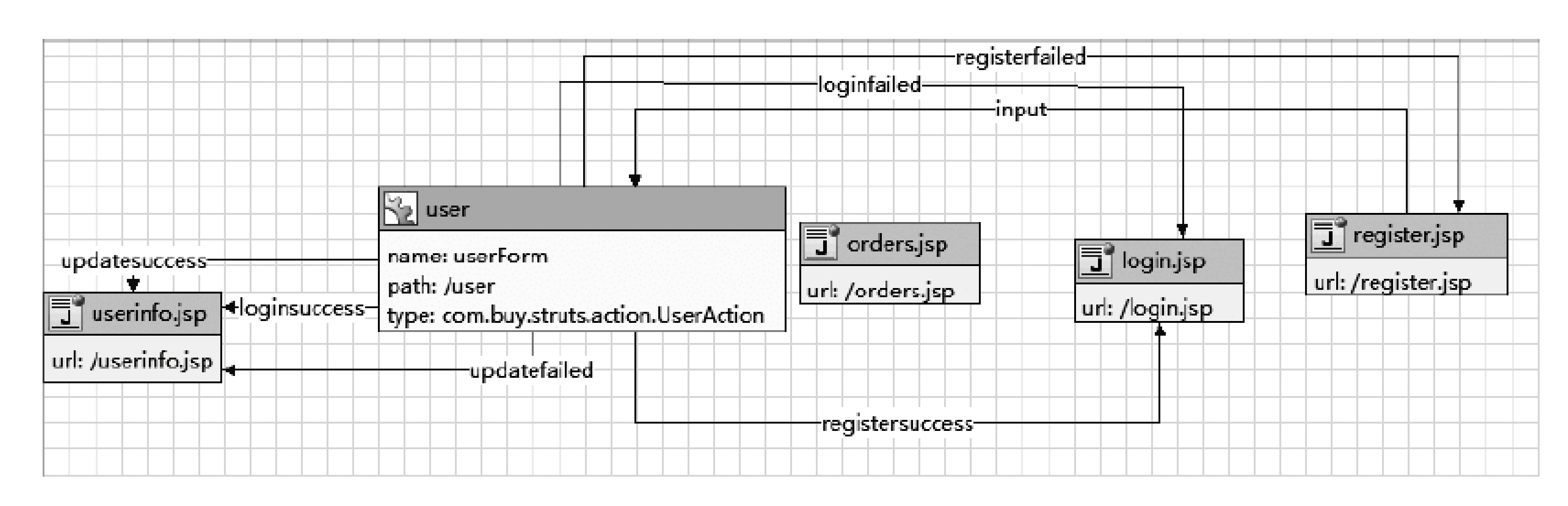


图 7-9 用户管理页面状态图

7.6.2 用户管理功能的模型层实现

1. 使用 Database Explorer 创建映射文件与持久化类

所有持久化类和映射文件，都存放在项目 BUY 的 com. buy. model 包中。具体步骤请参考第 5 章的 5. 4. 2 节。

1）查看结果——映射文件 UserTbl. hbm. xml

数据表 user_tbl 生成的映射文件为 UserTbl. hbm. xml，该文件实现了持久化类与数据表的映射，并配置了所有需要配置的元素和属性。其配置代码如代码清单 7-4 所示。

代码清单 7-4 映射文件 UserTbl. hbm. xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0
//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
    <class name="com.buy.model.UserTbl" table="user_tbl" catalog="buy">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="native" />
        </id>
        <property name="name" type="java.lang.String">
            <column name="name" length="100" not-null="true" />
```

```
        </property>
        <property name="password" type="java.lang.String">
            <column name="password" length="100" not-null="true" />
        </property>
        <property name="email" type="java.lang.String">
            <column name="email" length="200" not-null="true" />
        </property>
        <property name="msn" type="java.lang.String">
            <column name="msn" length="200" not-null="true" />
        </property>
        <property name="qq" type="java.lang.String">
            <column name="qq" length="20" not-null="true" />
        </property>
        <property name="tel" type="java.lang.String">
            <column name="tel" length="20" not-null="true" />
        </property>
        <property name="logtime" type="java.sql.Timestamp">
            <column name="logtime" length="19" not-null="true" />
        </property>
        <property name="coin" type="java.lang.Integer">
            <column name="coin" />
        </property>
        <property name="consume" type="java.lang.Integer">
            <column name="consume" />
        </property>
    </class>
</hibernate-mapping>
```

此时，你可以查看 hibernate.cfg.xml 文件，该文件也被自动添加了 UserTbl.hbm.xml 的配置。

```
<mapping resource="com/buy/model/beans/UserTbl.hbm.xml" />
```

2）产生结果——持久化抽象类 AbstractUserTbl.java

数据表 user_tbl 对应一个持久化的抽象类 AbstractUserTbl.java，该类包含了数据表 user_tbl 的列所对应的所有变量，并包含了 equals()和 hashCode()函数。完整的代码如代码清单 7-5 所示。

代码清单 7-5　持久化抽象类 AbstractUserTbl.java

```
package com.buy.model;

import java.sql.Timestamp;

/**
 * AbstractUserTbl entity provides the base persistence definition of the
```

```
 * UserTbl entity. @ author MyEclipse Persistence Tools
 * /

public abstract class AbstractUserTbl implements java.io.Serializable {

    //Fields

    private Integer id;
    private String name;
    private String password;
    private String email;
    private String msn;
    private String qq;
    private String tel;
    private Timestamp logtime;
    private Integer coin;
    private Integer consume;
    //Constructors
    /** default constructor * /
    public AbstractUserTbl() {
    }
    /** minimal constructor * /
    public AbstractUserTbl(String name, String password, String email,
            String msn, String qq, Timestamp logtime) {
        this.name= name;
        this.password= password;
        this.email= email;
        this.msn= msn;
        this.qq= qq;
        this.logtime= logtime;
    }
    /** full constructor * /
    public AbstractUserTbl(String name, String password, String email,
            String msn, String qq, String tel, Timestamp logtime, Integer coin,
            Integer consume) {
        this.name= name;
        this.password= password;
        this.email= email;
        this.msn= msn;
        this.qq= qq;
        this.tel= tel;
        this.logtime= logtime;
        this.coin= coin;
        this.consume= consume;
```

```
}

//Property accessors

public Integer getId() {
      return this.id;
}

public void setId(Integer id) {
      this.id= id;
}

public String getName() {
      return this.name;
}

public void setName(String name) {
      this.name= name;
}

public String getPassword() {
      return this.password;
}

public void setPassword(String password) {
      this.password= password;
}

public String getEmail() {
      return this.email;
}

public void setEmail(String email) {
      this.email= email;
}

public String getMsn() {
      return this.msn;
}

public void setMsn(String msn) {
      this.msn= msn;
}
```

```
    public String getQq() {
        return this.qq;
    }

    public void setQq(String qq) {
        this.qq= qq;
    }
public String getTel() {
        return this.tel;
    }

    public void setTel(String tel) {
        this.tel= tel;
    }

    public Timestamp getLogtime() {
        return this.logtime;
    }

    public void setLogtime(Timestamp logtime) {
        this.logtime= logtime;
    }

    public Integer getCoin() {
        return this.coin;
    }

    public void setCoin(Integer coin) {
        this.coin= coin;
    }

    public Integer getConsume() {
        return this.consume;
    }

    public void setConsume(Integer consume) {
        this.consume= consume;
    }

    /**
     * 判断相等性的方法,重写了 Object类中的方法
     * /
    public boolean equals(Object obj) {
        if (obj==null)
```

```
            return false;
        if (!(obj instanceof UserTbl))
            return false;
        UserTbl that= (UserTbl) obj;
        if (this.getId()==null||that.getId()==null)
            return false;
        return this.getId().equals(that.getId());
    }

    /**
     * 重写 Object 类的方法
     */
    public int hashCode(){
        if (this.hashCode()==0){
            int result=17;
            int idValue=this.getId()==null ?0 : this.getId().hashCode();
            result=result * 37+idValue;
        }
        return this.hashCode();

    }
}
```

3）产生结果——持久化类 User.java

AbstractUserTbl.java 是一个持久化类，因此，MyEclipse 还创建了一个实现类 UserTbl.java。MyEclipse 在创建持久化类时，分别创建了一个抽象类和实现类，目的是为了实现重用。该实现类的代码如代码清单 7-6 所示。

代码清单 7-6　持久化类 UserTbl.java

```
package com.buy.model;

import java.sql.Timestamp;

/**
 * UserTbl entity. @author MyEclipse Persistence Tools
 */
public class UserTbl extends AbstractUserTbl implements java.io.Serializable {

    //Constructors

    /** default constructor */
    public UserTbl(){
    }
```

```
    /** minimal constructor * /
    public UserTbl(String name, String password, String email, String msn,
            String qq, Timestamp logtime){
        super(name, password, email, msn, qq, logtime);
    }

    /** full constructor * /
    public UserTbl(String name, String password, String email, String msn,
            String qq, String tel, Timestamp logtime, Integer coin, Integer consume){
        super(name, password, email, msn, qq,tel, logtime, coin, consume);
    }

}
```

到目前为止，数据库表 user_tbl 对应的映射文件和持久化类都已经创建完毕了。

2. 编写 UserTblDAO. java

下面我们根据登录、注册中的业务逻辑来修改自动生成的 UsterTblDAO. java，该类也位于 com. buy. model 包中。如代码清单 7. 7 所示。

代码清单 7-7 UserTblDAO. java

```
package com.buy.model;

import java.sql.Timestamp;
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.LockMode;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.criterion.Example;
public class UserTblDAO extends BaseHibernateDAO {
    private static final Log log= LogFactory.getLog(UserTblDAO.class);
    //property constants
    public static final String NAME= "name";
    public static final String PASSWORD= "password";
    public static final String EMAIL= "email";
    public static final String MSN= "msn";
    public static final String QQ= "qq";
    public static final String TEL= "tel";
    public static final String COIN= "coin";
    public static final String CONSUME= "consume";

    public boolean update(UserTbl transientInstance){
```

```
        Session session=null;
        Transaction tx=null;
        boolean b=false;
        log.debug("saving UserTbl instance");
        try {
            session=getSession();                          //打开连接
            tx=session.beginTransaction();                 //开始事务
            session.update(transientInstance);             //保存实例到数据库
            tx.commit();
            b=true;
            log.debug("save successful");
        } catch (RuntimeException re){
            System.out.println(re.getMessage());
            b=false;
            log.error("save failed", re);
            throw re;
        }finally{
            if(tx!=null)
                tx.rollback();                             //事务回滚
            HibernateSessionFactory.closeSession();        //关闭连接
        }
        return b;
    }
    public boolean save(UserTbl transientInstance){
        Session session=null;
        Transaction tx=null;
        boolean b=false;
        log.debug("saving UserTbl instance");
        try {
            session=getSession();                          //打开连接
            tx=session.beginTransaction();                 //开始事务
            session.save(transientInstance);               //保存实例到数据库
            tx.commit();
            b=true;
            log.debug("save successful");
        } catch (RuntimeException re){
            System.out.println(re.getMessage());
            b=false;
            log.error("save failed", re);
            throw re;
        }finally{
            if(tx!=null)
                tx.rollback();                             //事务回滚
            HibernateSessionFactory.closeSession();        //关闭连接
```

```
        }
        return b;
    }
    public void delete(UserTbl persistentInstance){
        log.debug("deleting UserTbl instance");
        try {
            getSession().delete(persistentInstance);
            log.debug("delete successful");
        } catch (RuntimeException re){
            log.error("delete failed", re);
            throw re;
        }
    }

    public UserTbl findById(java.lang.Integer id){
        log.debug("getting UserTbl instance with id: "+ id);
        try {
            UserTbl instance= (UserTbl) getSession().get(
                    "com.buy.model.UserTbl", id);
            return instance;
        } catch (RuntimeException re){
            log.error("get failed", re);
            throw re;
        }
    }

    public List findByExample(UserTbl instance){
        log.debug("finding UserTbl instance by example");
        try {
            List results= getSession().createCriteria("com.buy.model.UserTbl")
                    .add(Example.create(instance)).list();
            log.debug("find by example successful, result size: "
                    + results.size());
            return results;
        } catch (RuntimeException re){
            log.error("find by example failed", re);
            throw re;
        }
    }

    public List findByProperty(String propertyName, Object value){
        log.debug("finding UserTbl instance with property: "+ propertyName
                + ", value: "+ value);
        try {
```

```
            String queryString= "from UserTbl as model where model."
                    + propertyName+ "= ?";
            Query queryObject= getSession().createQuery(queryString);
            queryObject.setParameter(0, value);
            return queryObject.list();
        } catch (RuntimeException re) {
            log.error("find by property name failed", re);
            throw re;
        }
    }

    public List findByName(Object name) {
        return findByProperty(NAME, name);
    }

    public List findByPassword(Object password) {
        return findByProperty(PASSWORD, password);
    }

    public List findByEmail(Object email) {
        return findByProperty(EMAIL, email);
    }

    public List findByMsn(Object msn) {
        return findByProperty(MSN, msn);
    }

    public List findByQq(Object qq) {
        return findByProperty(QQ, qq);
    }

    public List findByCoin(Object coin) {
        return findByProperty(COIN, coin);
    }

    public List findByConsume(Object consume) {
        return findByProperty(CONSUME, consume);
    }

    public List findAll() {
        log.debug("finding all UserTbl instances");
        try {
            String queryString= "from UserTbl";
            Query queryObject= getSession().createQuery(queryString);
```

```
            return queryObject.list();
        } catch (RuntimeException re) {
            log.error("find all failed", re);
            throw re;
        }
    }

    public UserTbl merge(UserTbl detachedInstance) {
        log.debug("merging UserTbl instance");
        try {
            UserTbl result= (UserTbl) getSession().merge(detachedInstance);
            log.debug("merge successful");
            return result;
        } catch (RuntimeException re) {
            log.error("merge failed", re);
            throw re;
        }
    }

    public void attachDirty(UserTbl instance) {
        log.debug("attaching dirty UserTbl instance");
        try {
            getSession().saveOrUpdate(instance);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }

    public void attachClean(UserTbl instance) {
        log.debug("attaching clean UserTbl instance");
        try {
            getSession().lock(instance, LockMode.NONE);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }

}
```

7.6.3 登录和注册功能的视图层实现

1. 编写 ActionForm Bean 类 UserForm

当用户进行登录或注册操作时，单击提交按钮之后，会由对应的 ActionServlet 接收该请求。用户提交的请求数据存放在 com. buy. struts. forms. UserForm 类的实例中。该类的代码如代码清单 7-8 所示。

代码清单 7-8 com. buy. struts. forms. UserForm. java

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package com.buy.struts.form;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

/**
 * MyEclipse Struts
 * Creation date: 11-09-2010
 *
 * XDoclet definition:
 * @struts.form name="userForm"
 */
public class UserForm extends ActionForm {
    /*
     * Generated fields
     */
    private int id;
    /** password property */
    private String password;

    /** email property */
    private String email;

    /** msn property */
    private String msn;

    /** qq property */
    private String qq;
    /** tel property */
```

```
private String tel;

/** name property */
private String name;

/*
 * Generated Methods
 */

/**
 * Method validate
 * @param mapping
 * @param request
 * @return ActionErrors
 */
public ActionErrors validate(ActionMapping mapping,
        HttpServletRequest request){
    //TODO Auto-generated method stub
    return null;
}

/**
 * Method reset
 * @param mapping
 * @param request
 */
public void reset(ActionMapping mapping, HttpServletRequest request){
    //TODO Auto-generated method stub
}

/**
 * Returns the password.
 * @return String
 */
public String getPassword(){
    return password;
}

/**
 * Set the password.
 * @param password The password to set
 */
public void setPassword(String password){
    this.password=password;
```

```
}

/**
 * Returns the email.
 * @return String
 */
public String getEmail(){
     return email;
}

/**
 * Set the email.
 * @param email The email to set
 */
public void setEmail(String email){
     this.email=email;
}

/**
 * Returns the msn.
 * @return String
 */
public String getMsn(){
     return msn;
}

/**
 * Set the msn.
 * @param msn The msn to set
 */
public void setMsn(String msn){
     this.msn=msn;
}

/**
 * Returns the qq.
 * @return String
 */
public String getQq(){
     return qq;
}

/**
 * Set the qq.
```

```
 * @param qq The qq to set
 * /
public void setQq(String qq){
     this.qq=qq;
}
/**
 * Returns the tel.
 * @return String
 * /
public String getTel(){
     return tel;
}

/**
 * Set the tel.
 * @param tel The tel to set
 * /
public void setTel(String tel){
     this.tel=tel;
}

/**
 * Returns the name.
 * @return String
 * /
public String getName(){
     return name;
}

/**
 * Set the name.
 * @param name The name to set
 * /
public void setName(String name){
     this.name=name;
}

public void setId(int id){
     this.id=id;
}

public int getId(){
     return id;
}
```

```
}
```

2. 编写用户登录页面 login.jsp

用户登录页面是一个带表单的 JSP 页面，该页面代码如代码清单 7-9 所示。

代码清单 7-9 login.jsp(不含美化代码)

```
<%@page language="java" pageEncoding="GBK"%>

<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html lang="true">
  <head>
    <html:base />

    <title>欢迎登录</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->

  </head>

  <body>
    <html:form action="user.do?type=doLog" method="post" focus="name">
      <table border="0">
        <tr>
          <td>用户名:</td>
          <td><html:text property="name" /></td>
        </tr>
        <tr>
          <td>密　码:</td>
          <td><html:password property="password" /></td>
        </tr>
        <tr>
          <td colspan="2" align="center"><html:submit value="登录" /></td>
```

```
        </tr>
      </table>
    </html:form>
  </body>
</html:html>
```

3. 编写用户注册页面 register.jsp

用户注册页面也是一个含有表单的 JSP 页面，该页面的代码如代码清单 7-10 所示。

代码清单 7-10 register.jsp

```
<%@page language="java" pageEncoding="GBK"%>

<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html lang="true">
  <head>
    <html:base />

    <title>欢迎注册</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->

  </head>

  <body>
    <html:form action="user.do?type=doRegister" method="post" focus="name">
      <table border="0">
        <tr>
          <td>用户名:</td>
          <td><html:text property="name" /></td>
        </tr>
        <tr>
          <td>密码:</td>
```

```
        <td><html:password property="password" /></td>
      </tr>
      <tr>
        <td>email:</td>
        <td><html:password property="email" /></td>
      </tr>
      <tr>
        <td>msn:</td>
        <td><html:text property="msn" /></td>
      </tr>
      <tr>
        <td>qq:</td>
        <td><html:text property="qq" /></td>
      </tr>
      <tr>
        <td>tel:</td>
        <td><html:text property="tel" /></td>
      </tr>

      <tr>
        <td colspan="2" align="center"><html:submit value="注册" /></td>
      </tr>
    </table>
  </html:form>
 </body>
</html:html>
```

4. 编写用户信息管理页面 userinfo.jsp

用户信息管理页面也是一个含有表单的 JSP 页面，该页面的代码如清单 7-11 所示。

代码清单 7-11　userinfo.jsp

```
<%@page language="java" pageEncoding="GBK"%>

<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html lang="true">
 <head>
  <html:base />

  <title>用户信息管理</title>
```

```
  <meta http-equiv="pragma" content="no-cache">
  <meta http-equiv="cache-control" content="no-cache">
  <meta http-equiv="expires" content="0">
  <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
  <meta http-equiv="description" content="This is my page">
  <!--
  <link rel="stylesheet" type="text/css" href="styles.css">
  -->

</head>

<body>
用户基本信息
  <html:form action="user.do?type=doUpdateInfo" method="post" focus="name">
    <table border="0">
    <tr>
    <td>用户名</td>
    <td><logic:present name="user">
          <html:hidden property="id" name="user" />
          <html:text name="user" property="name" disabled="true"/>
      </logic:present><logic:notPresent name="user">
          <input type="text" name="name" maxlength="50">
      </logic:notPresent></td>
    </tr>
    <logic:present name="user">
          <html:hidden property="password" name="user" />
      </logic:present>
      <tr>
        <td>Email:</td>
        <td><logic:present name="user">
          <html:text property="email" name="user" maxlength="50"/>
      </logic:present><logic:notPresent name="user">
          <input type="text" name="email" maxlength="50">
      </logic:notPresent></td>
      </tr>
      <tr>
        <td>MSN:</td>
        <td><logic:present name="user">
          <html:text property="msn" name="user" maxlength="50"/>
      </logic:present><logic:notPresent name="user">
          <input type="text" name="msn" maxlength="50">
      </logic:notPresent></td>
      </tr>
      <tr>
```

```
        <td>QQ:</td>
        <td><logic:present name="user">
          <html:text property="qq" name="user" maxlength="50"/>
      </logic:present><logic:notPresent name="user">
          <input type="text" name="qq" maxlength="50">
      </logic:notPresent></td>
      </tr>
    <tr>
        <td>TEL:</td>
        <td><logic:present name="user">
          <html:text property="tel" name="user" maxlength="50"/>
      </logic:present><logic:notPresent name="user">
          <input type="text" name="tel" maxlength="50">
      </logic:notPresent></td>
      </tr>
      <tr>
        <td colspan="2" align="center"><html:submit value="确认修改" /></td>
      </tr>
    </table>
</html:form>
```

用户密码修改

```
<html:form action="user.do?type=doUpdatePwd" method="post" focus=
"password">
    <table border="0">
      <tr>
        <td>原有密码:</td>
        <td><logic:present name="user">
          <html:hidden property="id" name="user" />

      </logic:present><html:password property="oldpassword" value=""/></td>
      </tr>
      <tr>
        <td>新密码:</td>
        <td><html:password property="password" value=""/></td>
      </tr>
      <tr>
        <td>确认密码:</td>
        <td><html:password property="repassword" value=""/></td>
      </tr>
      <tr>
        <td colspan="2" align="center">1<html:submit value="确认修改" /></td>
      </tr>
    </table>
```

```
    </html:form>
  </body>
</html:html>
```

7.6.4 用户管理功能的控制层实现

控制层主要任务是获取用户提交信息，并根据信息的处理结果转向对应的页面。主要过程是：

1. 开发 Action：创建用户处理控制类 UserAction.java。

该类是一个 DispatchAction，负责接收来自 login.jsp、register.jsp 和 userinfo.jsp 的请求，并根据请求参数 type 决定是进行登录、注册还是更新处理。功能的业务流程详见用户管理的逻辑设计章节。

- 该类中 doLogin()方法负责处理登录操作。

(1) 定义局部变量。forward 和 userForm。

(2) 从用户表单对象 userForm 中取得用户输入的用户名 name 和密码 password。

(3) 从 request 对象取得 Session 对象，判断用户 Session 对象是否为空，如果不为空，则需清空 Session 对象，再重新获得 Session 对象。

(4) 执行用户登录验证逻辑。

(5) 如果 isValid 为 true，则在 Session 中保存一个变量 Constants.USERNAME_KEY；如果 isValid 为 false，则在 errors 中保存一条错误消息 login.error.faild。

(6) 判断 errors 中是否有错误消息，如果不为空，则调用父类的方法 saveErrors()来将 errors 对象保存在 request 中，并通过 mapping 对象获得登录失败 loginfail 所指向的 ActionForward 对象；如果错误消息为空，则通过 maping 对象获得 loginsucess 所指向的 ActionForward 对象。

(7) 返回 ActionForward 对象 forward。

- doRegister()方法负责处理注册操作。

(1) 定义三个局部变量 errors、forward 和 userForm。

(2) 从用户表对象 userForm 中取得用户输入的用户名 name、密码 password、email、msn、qq 等信息。

(3) 执行用户注册检查逻辑。

(4) 如果 isRegister 为 true，则表示不允许注册，在 errors 中保存一条错误消息 register.error.failed。

(5) 判断 errors 中是否有错误消息，如果不为空，再调用父类的 saveErrors()方法来将 errors 对象保存在 request 中，并通过 mapping 对象取得注册失败 registerfail 所指向的 ActionForward 对象；如果错误消息为空，则通过 mapping 对象取得注册成功 registersuccess 所指向的 ActionForward 对象。

(6) 返回 ActionForward 对象 forward。

- doUpdateInfo()方法负责处理信息更改操作。

(1) 定义局部变量 forward 和 userForm。
(2) 执行用户更新逻辑。
(3) 如果更新成功,则保存最新数据到 Session 中。
(4) 返回 ActionForward 对象 forward。
代码如代码清单 7-12 所示。

代码清单 7-12 UserAction.java

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package com.buy.struts.action;

import java.util.Calendar;
import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import com.buy.model.UserTbl;
import com.buy.model.UserTblDAO;
import com.buy.struts.form.UserForm;

/**
 * MyEclipse Struts
 * Creation date: 11-09-2010
 *
 * XDoclet definition:
 * @struts.action path="/user" name="userForm" input=
   "/register.jsp" parameter="type" scope="request" validate="true"
 * @struts.action-forward name="loginsuccess" path="/orders.jsp"
 * @struts.action-forward name="registersuccess" path="/login.jsp"
 * @struts.action-forward name="registerfailed" path="/register.jsp"
 * @struts.action-forward name="loginfailed" path="/login.jsp"
 */
public class UserAction extends DispatchAction {
    /*
     * Generated Methods
```

```
     * /

   /**
     * Method execute
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @return ActionForward
     * /
   public ActionForward doLogin(ActionMapping mapping, ActionForm form,
           HttpServletRequest request, HttpServletResponse response){
       UserForm userForm= (UserForm) form;
                                       //将方法的输入变量强制转换为 UserForm类型的表单对象
       ActionForward forward= new ActionForward();
                                       //请求转发对象,用于请求成功或失败时进行转发
       try{
           //检查 Session 是否有效
           HttpSession session= request.getSession(false);
           if(session!= null)
               session.invalidate();
           //创建一个新的 Session
           session= request.getSession(true);
           //验证登录
           boolean isValid= false;
           UserTbl user= valid(userForm);
           if(user!= null)
               isValid= true;
           if(isValid){
               session.setAttribute("user", user);
               forward= mapping.findForward("loginsuccess");
           }
           else
               forward= mapping.findForward("loginfailed");
       }catch(Exception e){
           forward= mapping.findForward("loginfailed");
       }
       return forward;
   }
   /**
     * 处理注册请求
     * @param mapping
     * @param form
     * @param request
```

```
     * @param response
     * @return
     * /
    public ActionForward doRegister(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response){
        UserForm userForm= (UserForm) form;
                                //将方法的输入变量强制转换为 UserForm类型的表单对象
        ActionForward forward= new ActionForward();
                                //请求转发对象,用于请求成功或失败时进行转发
        try{
            //验证用户名是否存在
            boolean isExist= false;
            if(isExist(userForm.getName()))
                isExist= true;
            if(isExist){
                forward=mapping.findForward("registerfailed");
            }
            else{
                add(userForm);                    //执行用户注册
                forward=mapping.findForward("registersuccess");
            }
        }catch(Exception e){
            forward= mapping.findForward("registerfailed");
        }
        return forward;
    }
/* 更新用户信息 * /
public ActionForward doUpdateInfo(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response){
        UserForm userForm= (UserForm) form;
                                //将方法的输入变量强制转换为 UserForm类型的表单对象
        ActionForward forward= new ActionForward();
                                //请求转发对象,用于请求成功或失败时进行转发
        try{
                updateInfo(userForm);             //执行用户信息更改
                forward=mapping.findForward("updatesuccess");
                HttpSession session= request.getSession();
                UserTbl user= valid(userForm);
                session.setAttribute("user", user);
        }catch(Exception e){
            forward= mapping.findForward("updatefailed");
        }
        return forward;
    }
```

```
/**
 * 用户注册业务实现
 * @param userForm
 */
private void add(UserForm userForm){
    UserTbl user=new UserTbl();
    UserTblDAO userDAO=new UserTblDAO();
    //初始化 UserTbl 对象 user
    user.setName(userForm.getName());
    user.setPassword(userForm.getPassword());
    user.setMsn(userForm.getMsn());
    user.setQq(userForm.getQq());
    user.setTel(userForm.getTel());
    user.setEmail(userForm.getEmail());
    user.setCoin(0);
    user.setConsume(0);

    userDAO.save(user);

    }
/**
 * 用户信息更改业务实现
 * @param userForm
 */
private void updateInfo(UserForm userForm){
    UserTbl user=new UserTbl();
    UserTblDAO userDAO=new UserTblDAO();
    //初始化 UserTbl 对象 user
    user.setId(userForm.getId());
    user.setMsn(userForm.getMsn());
    user.setQq(userForm.getQq());
    user.setTel(userForm.getTel());
    user.setEmail(userForm.getEmail());
    user.setPassword(userForm.getPassword());
    userDAO.update(user);

    }
private boolean isExist(String name){
    UserTbl user=new UserTbl();
    UserTblDAO userDAO=new UserTblDAO();
    List list=userDAO.findByName(name);
    if(list!=null && list.size()!=0)
        return true;
    else
```

```
            return false;
        }
    /**
     * 验证用户是否能登录
     * @param userForm
     * @return
     */
        private UserTbl valid(UserForm userForm){
            UserTbl user=new UserTbl();
            UserTblDAO userDAO=new UserTblDAO();

            user.setName(userForm.getName());
            user.setPassword(userForm.getPassword());

            List list=userDAO.findByExample(user);
            if(list!=null&&list.size()>0)
                return (UserTbl)list.get(0);
            else return null;
        }
    }
```

2. 在 struts-config. xml 文件中配置<action>标签，处理用户的登录和注册操作。代码如代码清单 7-13 所示。

代码清单 7-13 struts-config. xml 中有关 UserAction 的配置

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-
config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans>
    <form-bean name="userForm" type="com.buy.struts.form.UserForm" />

  </form-beans>

  <global-exceptions />
  <global-forwards />
  <action-mappings>
    <action
      attribute="userForm"
      input="/register.jsp"
      name="userForm"
      parameter="type"
```

```
        path="/user"
        scope="request"
        type="com.buy.struts.action.UserAction">
        <forward name="loginsuccess" path="/userinfo.jsp" />
        <forward name="registersuccess" path="/login.jsp" />
        <forward name="registerfailed" path="/register.jsp" />
        <forward name="loginfailed" path="/login.jsp" />
        <forward name="updatesuccess" path="/userinfo.jsp" />
        <forward name="updatefailed" path="/userinfo.jsp" />
      </action>

    </action-mappings>

    <message-resources parameter="com.buy.struts.ApplicationResources" />
</struts-config>
```